VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

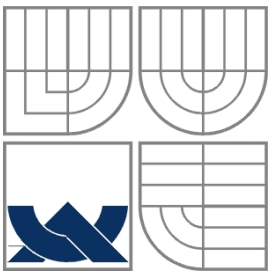
KARETNÍ HRA TAROKY PRO MOBILNÍ ZAŘÍZENÍ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

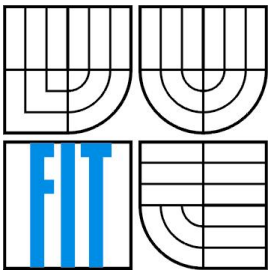
AUTOR PRÁCE
AUTHOR

Bc. Vít Sykala

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

KARETNÍ HRA TAROKY PRO MOBILNÍ ZAŘÍZENÍ

CARD GAME TAROT FOR MOBILE DEVICES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Vít Sykala

VEDOUCÍ PRÁCE
SUPERVISOR

Křivka Zbyněk, Ing., Ph.D.

BRNO 2010

Abstrakt

Tato diplomová práce se zabývá vývojem hry Taroky pro mobilní zařízení. Hra je vyvíjena v programovacím jazyce Java – J2ME. Aplikace umožňuje hrát Taroky jednomu až čtyřem lidem. Inteligence hráčů reprezentovaných mobilním zařízením je implementovaná jako expertní systém. Pravidla expertního systému je možno změnit bez nutnosti rekompilace celé aplikace. Navržený a implementovaný expertní systém využívá jako bázi znalostí odlehčenou verzi jazyka Prolog. Pravidla tohoto systému jsou ve tvaru přesně definovaném touto prací.

Abstract

This thesis describes development of the Tarot game for mobile devices. The game is developed in Java - J2ME. The application allows to play Tarot to one through four people. The player intelligence represented by the mobile device is implemented as an expert system. The expert system rules can be changed without necessity to recompile the application. Designed and implemented expert system uses a lightweight version of Prolog language as a knowledge base. The system requires the rules of precisely defined form specified in this thesis.

Klíčová slova

J2ME, Taroky, Expertní systém

Keywords

J2ME, Tarot, Expert system

Citace

Vít Sykala: Karetní hra Taroky pro mobilní zařízení, Brno, FIT VUT v Brně, 2010

Karetní hra Taroky pro mobilní zařízení

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing Zbyňka Křivky, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Vít Sykala
(20.5.2010)

Poděkování

Chtěl bych tímto poděkovat svému vedoucímu práce za příkladné vedení, všestrannou pomoc a nasměrování správným směrem.

Obsah

1 Úvod

V dnešní době vývoj mobilních aplikací zažívá rozkvět. Vzpomeňme si na situaci před čtrnácti lety, kdy bylo vlastnictví mobilních telefonů velice prestižní záležitostí. V roce 1996 stál nejlevnější telefon zhruba 10 000 Kč. A mezi běžnými lidmi bylo nemyslitelné, že by jejich mobilní telefon mohl zvládat více než volání a posílání textových zpráv. V dnešní době je již běžným standardem v mobilním telefonu mít mnoho aplikací a služeb. Nabídka mobilních aplikací je opravdu široká. Množství mobilních telefonů dnešní doby používá operační systém, ať už některou verzi *Windows Mobile*, *Symbian* nebo jiný, většinou linuxového typu (Android, Openmoko, atd.). Trend vývoje mobilních telefonů je analogií vývoje počítačů. Podobně i aplikace a přístupy, známé původně z osobních počítačů, jsou stále častěji používány i v oblasti softwarového vývoje pro mobilní zařízení.

Podobně i tato práce se pokouší vyvinout umělou inteligenci pro hraní hry Taroky v mobilním telefonu. Jsou zde proto zkoumány přístupy používané dnes v počítačích a snažím se navrhnout a implementovat expertní systém pro hraní hry taroky na mobilním zařízení.

1.1 Cíl práce

Cílem této diplomové práce je prostudovat detailně pravidla karetní hry Taroky. Následně pak vybrat vhodnou platformu pro tvorbu aplikací pro mobilní zařízení. Prozkoumat možnosti expertních systémů a případně i dalších alternativ pro tvorbu inteligentního rozhodování při hraní této hry. Navrhnout aplikaci využívající získaných poznatků tak, aby se všechny tři části daly skloubit do jedné ucelené aplikace. Navržené řešení implementovat pro vybranou platformu.

1.2 Přehled kapitol

Druhá kapitola nazvaná Karetní hra taroky se zabývá popisem pravidel této karetní hry. V kapitole nejde o detailní vysvětlení hry, nýbrž o uvedení do problematiky hry samotné. Zde také uvádím odkazy na relevantní pravidla, ze kterých je při implementaci implicitně počítáno. Po přečtení a pochopení této kapitoly by měl čtenář být schopen pozorovat hru a chápat, o čem se v ní jedná.

Třetí kapitola se zabývá technologickými požadavky na aplikace pro mobilní zařízení. Jak již název napovídá, jde o programování v J2ME. Tato platforma je určená přímo pro mobilní zařízení a v dnešní době je velice rozšířená a proto jsem se pro danou platformu rozhodl. Dalším důvodem pro tuto volbu je bezesporu programování v jazyce Java, které je příjemné a mám s ním dobré zkušenosti.

V dnešní době známe několik přístupů k řešení umělé inteligence, mezi které patří: **Expertní systémy** *(ES)*, **Umělé neuronové sítě**, vhodné zejména pokud neznáme pravidla, podle kterých se inteligentní systém v daných situacích rozhoduje, případně pokud je tyto pravidla náročné získat, **Genetické algoritmy**, které vytvářejí řádově stovky náhodných jedinců, z nichž vybírají ty úspěšné, jejichž části kombinují k nalezení ještě úspěšnějších jedinců. Čtvrtá, asi nejdůležitější z teoretických kapitol je věnována expertním systémům. Jsou zde rozebrány výhody a nevýhody jednotlivých přístupů, základní klasifikace expertních systémů a způsoby návrhu. V závěru této kapitoly

je proveden základní návrh tvaru pravidel pro ES a způsob, jakým budou pravidla získána. Také se zde zmiňuji o způsobech hodnocení úspěšnosti jednotlivých pravidel.

Pátá kapitola o návrhu a implementaci aplikace samotné. Velká část této kapitoly se věnuje uživatelskému rozhraní aplikace a částečně už je zde naznačeno i rozhraní expertního systému a případných dalších typů hráčů, kdyby byla vyvíjena další verze aplikace.

Detailně je problematika rozhraní expertního systému popsána v kapitole šest. Po přečtení této kapitoly by měl člověk být schopen programovat pravidla v jazyce expertního systému. V této kapitole jsou dále rozebrány problémy použitého interpretu jazyka Prolog.

Poslední kapitola se zabývá konkrétní tvorbou pravidel pro expertní systém. Jsou zde uvedeny základní konstrukce jazyka expertního systému v konkrétních příkladech použití.

2 Karetní hra Taroky

Karetní hra Taroky je jednou z nejstarších karetních her. První výskyt tarokových karet, které se ale výrazně lišily od těch dnešních, se datuje již do první poloviny 15. století do okolí italského města Ferrari. Později se tyto karty rozšířily i na území tehdejšího Rakousko-Uherska. V 18. století se, tehdy ještě 78 listů bohatě zdobených, tarokových karet začalo využívat i pro okultní a mystické účely. Dodnes slyšíme o věštbě z karet. Taroky jsou asi nejstarší karetní hrou hranou na území současné České republiky, hlavně tedy na Moravě. Dokonce je starší než Mariáš a Bridge [1], kterým se původ datuje někdy do 18. století.

Tato karetní hra se řadí mezi takzvané *zdvihové hry* (zdvih je zde balíček karet, který hráč vezme v daném kole) podobně jako třeba hry Mariáš a Tisíc. V současné době se v ČR hraje jako hra čtyř hráčů s padesáti čtyřmi kartami v balíčku. V zahraničí se například hraje s jiným množstvím karet a v minulosti se i u nás hrála varianta pro 3 hráče [2]. Tato hra je velice variabilní a téměř každá skupina hráčů má svá vlastní modifikovaná pravidla. Pravidla jednotlivých skupin se liší někdy jen drobnými odchylkami v hodnocení jednotlivých hlášek, někdy ovšem zcela zásadní změnou hry, někdy jen nepodstatnými, ale hráči striktně dodržovanými pravidly pro rozdávání.

Po přečtení prvních pravidel přiložených ke kartám (viz příloha Pravidla hry „Taroky“) a po odehrání několika her s různými lidmi, kteří taroky hrají již léta, jsem začal hledat pravidla, která by byla vhodná vzít jako referenční pro mou aplikaci. Nakonec jsem nalezl pravidla [3] obsahující většinu podstatných herních principů detailně vysvětlených, které budu používat v mé aplikaci a podle kterých budou níže popsána pravidla hry taroky.

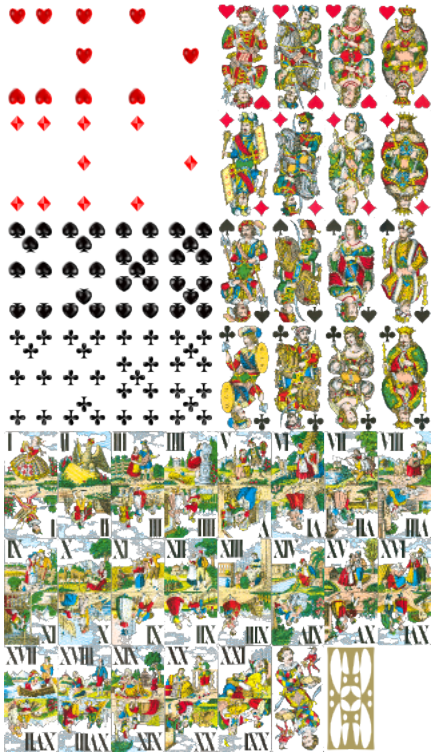
2.1 Pravidla hry

Pro hraní karetní hry Taroky se u nás, jak již bylo řečeno, používají tarokové karty s padesáti čtyřmi kartami. Tyto karty obsahují čtyři barvy po osmi kartách a 22 takzvaných taroků, které jsou stálými trumfy. Na obrázku 1 vidíme všech 54 tarokových karet a rubovou stranu karet. V prvních 4 řádcích jsou jednotlivé barvy (srdce neboli herce, káry, piky a kříže) vždy seřazeny vzestupně podle hodnoty karty. Následují 3 řádky trumfů značené římskými čísly. Karta bez označení nazývaná *Škýz* je nejvyšším trumfem a v některých kartách je označována XXII. Dalšími pojmenovanými kartami jsou Tarok číslo XXI *Mond* a tarok číslo I *Pagát*.

Každá hra se začíná rozdáním karet. Hráči obdrží 12 karet a 6 karet zůstane položených v balíčku na stole (říká se mu talón) tak, aby žádný z hráčů nevěděl, jaké jsou to karty. V rozdávání je skryto mnoho nejrůznějších pravidel. Je přesně specifikováno, kolikáté karty jsou v talonu a po kolika kartách se rozdává. Někdy se vybírá pořadí karet a mnoho dalších možností, které pro hru samotnou nejsou nijak podstatné. Jde jen o jakési zpestření jinak nudného rozdávání, proto se tímto nebudu zabývat.

Po rozdání karet začíná dražba hry, kterou začíná předák (hráč sedící po pravici rozdávajícího) a následně v pořadí protisměru hodinových ručiček i všichni ostatní hráči. Draží se jedna z následujících her (seřazeno vzestupně): první povinnost, druhá povinnost, Preferanc, sólo. První hráč povinně draží minimálně první povinnost. Každý další má dvě možnosti: souhlasit s nejvyšší vydraženou hrou nebo dražit hru vyšší. Hraje se nejvyšší vydražená hra. Hráč, který ji vydražil,

se jmenuje vydražitel. Pokud všichni souhlasí se základní povinnosti, předák má možnost vyhlásit speciální typ hry Varšava.



obrázek 1: Tarokové karty

Po dražbě následuje rozebrání balíčku dle typu hry a následně je čas na hlášky. Hlášky dělíme na prozrazující a zavazující.

Zavazující hlášky jsou dvě Pagát a Valát. Hlášením hlášky Pagát (hlášená za 20b., nehlášená za 10b.) se hráč zavazuje vzít poslední zdvih Pagátem. To předpokládá uchování Pagáta do posledního kola a zároveň donucení ostatních hráčů použít všechny trumfy před posledním kolem. Hlášením hlášky Valát (hlášená za 140b., nehlášená za 70b.) se hráč zavazuje sebrat všechny zdvihy ve hře.

Prozrazujících hlášek je více a jsou uvedeny v následujícím přehledu. Většina pravidel vyžaduje tyto hlášky hlásit, jsou tedy povinné.

Trul.....taroky I, XXI, XXII.....5b.
honery.....4 x pětibodová karta.....5b.
Královské honery.....4 x král.....10b.
Taroky.....10-12 taroků.....10b.
Taročky.....8-9 taroků.....5b.
Barvy.....0 taroků.....10b.
Barvičky.....1-2 taroky.....5b.

Tento výčet není úplný, existují ještě *Trulhonery*, hlášené hráčem majícím Trul a alespoň jednoho krále k tomu. Bodové hodnocení je poté součtem hodnocení za hlášku trul a hlášku honery, tedy 10b.

Flek, kontra flek a super flek jsou hlášeny po hláškách a vždy zdvojnásobují cenu toho, na co jsou uděleny. Udělit flek lze na hlášky Pagát a Valát nebo celou hru. Podobně jako je tomu

v jiných hrách, flek může udělit kdokoli z hráčů. Kontra flek (občas nazývaný reflek), uděluje hráč hrající danou hru nebo hlášku a super flek opět jeden z ostatních hráčů. Body je takto tedy možno násobit až osmi.

Samotnou hru vždy začíná předák, ať už ji vydražil kdokoli vynesením karty v prvním kole. První karta v každém kole určuje barvu kola, ostatní hráči postupně v protisměru hodinových ručiček vynášejí karty. Platí povinnost přiznat barvu (tedy pokud mám na ruce barvu stejnou jako barva kola, musím dát danou barvu). Pokud nemám barvu, tak musím dávat trumf a pokud nemám ani trumf, mohu vynést libovolnou kartu z ruky. Každé kolo (zdvih) vezme hráč s nejvyšší kartou v kole. Nejvyšší karta v kole je nejvyšší použitý trumf. Pokud trumfy nejsou použity, tak je jí nejvyšší použitá karta v barvě kola. Hraje-li se Varšava, platí navíc povinnost přebíjet. Pokud tedy mám barvu vyšší hodnoty, tak ji použiji. Totéž platí, pokud barvu nemám a mohu přebít trumfem, tak to musím udělat.

Následně se sečtou sebrané karty jednotlivých týmů a hra jako celek je vyhodnocena (viz 2.1.2). Většinou se taroky hrají jako hazardní hra. Po vyhodnocení hry hráči tedy zaplatí pohledávky ostatním hráčům. Podle předem domluveného pravidla se posune povinnost míchat karty a je možno zahájit další hru.

2.1.1 Druhy her

Hru **Základní povinnost** vydražitel zahájí slovy: „volám devatenáctku“, čímž říká, že bude hrát s hráčem, který má na ruce tarok číslo XIX. Pokud by vydražitel měl tarok XIX, volá osmnáctku a tak dále. Nejnižší tarok, který může být takto volán, je tarok číslo XVI a nižší taroky není možno volat. Vydražitel vezme první čtyři karty z talónu, hráč po jeho levici další kartu a hráč sedící naproti vydražiteli poslední kartu. Zde některá pravidla umožňují a jiná zakazují vzdát hru, pokud by vydražitel vzal z talónu kartu, kterou volal (tedy hraje sám se sebou proti ostatním). Každý z hráčů opět na stůl před sebe odloží tolik karet, kolik si vzal z talónu, a tyto karty jsou připočteny ke kartám, které v jednotlivých kolech vezme. Tímto způsobem je zakázáno odložit jakoukoli pětibodovou kartu (král, Mond, Štýz, Pagát) nebo tarok. Tarok může být takto odložen jen v případě, kdy má hráč na ruce jen taroky a krále. V takovém případě se tarok odkládá tak, aby jej všichni hráči viděli, tedy lícem vzhůru.

Druhá povinnost je úplně stejná jako základní povinnost s tím rozdílem, že vydražitel se zavazuje vzít poslední kolo Pagátem (nejnižší trumf ve hře).

Preferanc, občas nazýván *trojka*, je hra, při níž vydražitel hraje sám proti všem ostatním hráčům. Na začátku vezme vydražitel s talónu první 3 karty, podívá se na ně, a pokud chce, vezme si je a odloží jiné tři karty (podobně jako je to u první povinnosti). V takovém případě se jedná o trojku s prvními kartami. Další možností je ukázat tyto tři karty protihráčům a vzít si zbylé tři karty z talónu. V tomto případě se hra násobí dvěma, jedná se o trojku s druhými kartami. Poslední možností je vzít si původní tři karty. V takovém případě se hra násobí třemi a jedná se o takzvanou trojku s třetími kartami. Zbylé tři karty připadnou vždy protihráčům, kterým se rovnou přidají k jejich sebraným kartám.

Sólo je nejvyšší hra. Hráč, který ji vydražil, hraje sám proti všem ostatním. Talón se nijak nedělí a připadá automaticky celý protihráčům.

Varšava - tuto hru vyhlašuje pouze předák. Může tak učinit, jen pokud mu všichni odsouhlasili základní povinnost. V této hře na rozdíl od ostatních jde o to nahrát co možná nejméně bodů a platí

povinnost přebíjet. Talon dostane na starosti předák a ke každému z prvních šesti kol z něj přihodí jednu kartu. Tato karta nijak neovlivňuje to, kdo dané kolo sebere, pouze je přidána jako bodová zátěž daného kola.

2.1.2 Hodnocení hry

Body z karet jsou vypočteny podle vzorce 2.1, kde je *z* rovno 1, pokud počet karet není dělitelný třemi, jinak je rovno 0. Dělení je v tomto vzorci chápáno jako celočíselné. Součet bodů v obou týmech je dohromady vždy 70. Výhra týmů je spočtena podle vzorce 2.2 (záporná výhra je prohra).

$$body = \sum(hodnotKaret) - (početKaret\ MOD\ 3) * 2 - z$$

vzorec 2.1: výpočet bodu za sebrané karty

$$výhra\ týmu = (body - 35) * násobení\ hry + uhrané\ hlášky - hlášky\ neuhrané$$

vzorec 2.2: výpočet výhry jednotlivých týmů

Bodové hodnoty karet jsou uvedeny v následujícím přehledu:

5b.....Král, taroky číslo I, XXI, XXII
4b.....Dáma
3b.....Kaval neboli jezdec
2b.....Kluk
1b.....Jakkoliv jiná karta

Pokud je hrána varianta hry Varšava, tak se způsob hodnocení hry v jednotlivých pravidlech velice liší. Zde popsaná varianta vychází z pravidel popsaných na internetu (viz [3]). Hráč, který při této hře získal nejvíce bodu, prohrál. Kdo prohrál, je povinen každému z ostatních hráčů dát příslušný počet bodů, podle toho kolik prohrál. Počty bodů v závislosti na bodech za sebrané karty jsou následující.

Bodová hodnota sebraných karet	**Počet bodu k vyplacení**
(0, 29>	10
<30, 39>	20
40 a více	40

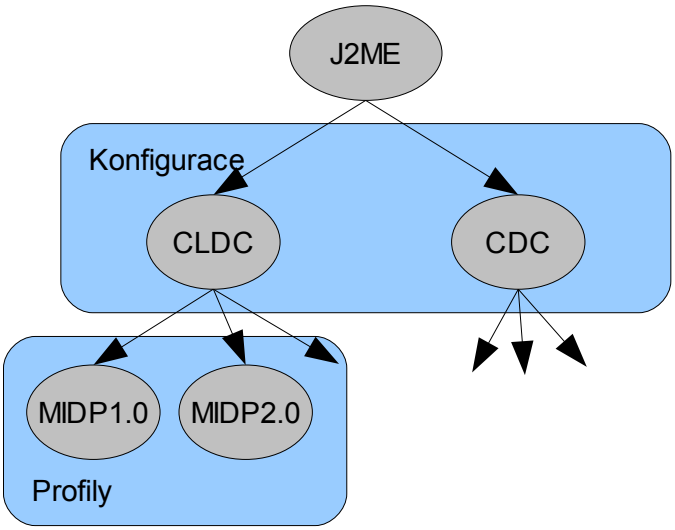
Při tomto vyplácení navíc hráči, kteří nesebrali žádnou kartu, jsou vyplaceni dvojnásobně. Tedy pokud by někdo například sebral všechny karty, tak by každému hráči byl povinen dát osmdesát bodů. Pokud více hráčů prohraje, tak se o povinnost vyplatit ostatní hráče rovnoměrně rozdělí.

Příklad: hráči A, B, C, D počítají výsledky hry Varšava. Hráči A a B vzali karty za třicet bodů každý a prohráli. Hráč C nevzal žádnou kartu. Hodnocení bude následující: Hráč C dostane dvacet bodů od hráče A a dvacet bodů od hráče B. Hráč D dostane deset bodů od hráče A a deset bodů od hráče B [3].

3 J2ME

Distribuce Javy pro mobilní zařízení zvaná Java 2 Micro Edition označována J2ME je určena pro vývoj aplikací pro mobilní zařízení. Tato platforma je široce podporována většinou výrobců těchto zařízení. Vývoj J2ME je zajištěn projektem Java Community Process (JCP) [4]. V tomto projektu je zapojeno více než 600 společností, mezi které patří například [5]: Nokia Corporation, Nokia Siemens Networks GmbH & Co. KG, Symbian Ltd, atd.

Díky podpoře (například v NetBeans) a rozšířenosti této mobilní platformy jsem se rozhodl pro její použití. Platforma J2ME samozřejmě není jediným standardem. Jednotlivá zařízení se mohou velice lišit svým výkonem a velikostí paměti nebo rozlišením, barevnou hloubkou obrazovky a mnoha dalšími parametry. Proto existují takzvané konfigurace a profily, které jsou dobře vysvětleny v [6]. Zde chci zdůraznit pouze ty, které jsem se rozhodl používat. Na obrázku 2 je vidět vzájemný vztah konfigurace, profilu a J2ME.



obrázek 2: Vzájemný vztah J2ME konfigurace a profilů

3.1 Konfigurace CLDC (Connected Limited Device Configuration)

Tato konfigurace je určena zejména pro zařízení s velmi omezenými zdroji, malou pamětí a ne příliš výkonným procesorem. CLDC dále definuje minimální požadované knihovny, které koncová implementace musí nabízet. Tato konfigurace nenařizuje koncové implementaci použití žádného konkrétního virtuálního stroje. Společnost Sun nabízí referenční virtuální stroj KVM (Kilobyte Virtual Machine). KVM byl vytvořen jako nejmenší možná kompletní implementace Virtuálního stroje pro Javu [7].

Minimální celková paměť pro běh aplikace je 160kB až 512kB. Každé zařízení CLDC by mělo mít minimálně 128kB stálé paměti pro virtuální stroj a 32kB dočasné paměti využitelné pro práci virtuálního stroje. To vše by mělo být nezávisle na jiných aplikacích [8].

Procesor zařízení musí být 16-bitový nebo 32-bitový s minimální taktovací frekvencí 25MHz.

Tato zařízení by měla být připojitelná k některému druhu sítě. Podle JSR139 musí podporovat následující knihovny [9]:

- **java.io** Vstupně výstupní operace a streamy.
- **java.lang** Základ pro programování v jazyce Java.
- **java.lang.ref** Podpora slabé reference.
- **java.util** Práce s kolekcemi, datem a časem.
- **javax.microedition.io** Framework pro připojování k zařízením.

Vzhledem k tomu, že specifikace CLDC vznikla v květnu roku 2000 [10], technologie už od té doby pokročila. V dnešní době mobilní telefony mají běžně procesory s frekvencemi v řádu stovek MHz. Podobná situace je také ve velikosti dostupné paměti pro běh aplikací, rozlišení a barevné hloubce obrazovky.

3.2 Profily MIDP (Mobil Information Device Profile)

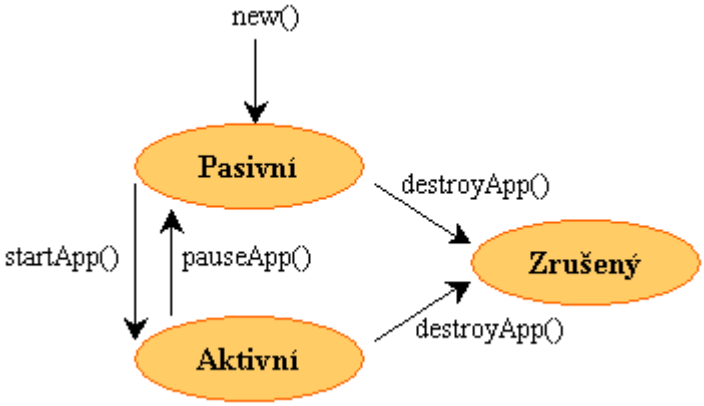
Tento standard vznikl pro bezdrátová zařízení, například pro mobilní telefony[8]. Stejně jako konfigurace CLDC je spravován JCP. Tento standard definuje další minimální parametry zařízení, zejména s ohledem na tvorbu uživatelského rozhraní, správy běhu aplikace, ukládání perzistentních dat a připojitelnosti k síti. Konkrétně se jedná o následující balíčky:

- **javax.microedition.lcdui** Komponenty pro práci s uživatelským rozhraním
- **javax.microedition.midlet** Řízení běhu aplikace, podle tohoto se aplikacím pracujícím nad MIDP většinou říká „MIDlet“
- **javax.microedition.rms** Ukládání perzistentních dat

MIDP ve verzi 2.0 dále definuje minimální rozlišení obrazovky 96 x 54 pixelů, vstupní zařízení jako klávesnici, případně stylus, a rozšiřuje paměťové požadavky o nutnost mít minimálně 8kB paměti pro perzistentní data[8], min 128kB dočasné paměti a minimálně 256kB paměti pro virtuální stroj. Dále obsahuje podporu pro snazší vývoj her (celoobrazový režim, herní tlačítka, atd.), připojitelnost zařízení [11] a možnost pracovat s plovoucí řádovou čárkou, což předchozí verze MIDP1.0 neměla.

3.3 Programování MIDletů

Stejně jako při programování appletu, jehož hlavní třída je zděděna od třídy `java.applet.Applet`, se v aplikaci J2ME při programování pomocí MIDP hlavní třída dědí od třídy `javax.microedition.midlet.MIDlet`. Těmto aplikacím se říká MIDlety. Podobně jako applet má i MIDlet svůj životní cyklus (viz obrázek 3), se kterým je třeba při návrhu aplikace počítat. Důležité je vyřešit problém přerušení aplikace například příchozím hovorem, aby se po návratu aplikace dala spustit a pokračovala od posledního přerušení.



obrázek 3: Životní cyklus MIDletu

4 Expertní systém

Je systém, který na základě expertních znalostí a odvozovacích pravidel nalezne řešení problému v dané oblasti. Mezi hlavní výhody expertního systému patří oddělení báze znalostí od řídícího mechanizmu, díky čemuž můžeme samotnou inteligenci snadno vyměnit. V dnešní době se expertní systémy často používají jako subsystémy větších celků. Tyto systémy jsou navrhovány na míru pro danou oblast zkoumání. Na expertní systémy jsou často kladeny požadavky zejména v oblasti práce s neurčitostí. Jiná definice expertního systému říká: *„Expertní systémy jsou počítačové programy simulující rozhodování experta při řešení složitých úloh a využívající vhodně zakódovaných, explicitně vyjádřených speciálních znalostí, převzatých od experta, s cílem dosáhnout ve zvolené problémové oblasti kvality rozhodování na úrovni experta“* [12].

Expertní systémy můžeme rozdělit na takzvané diagnostické a plánovací. **Diagnostické** expertní systémy jsou používány například ve zdravotnictví, kde podle příznaků mohou pomoci lékařům určit nemoc pacienta, případně nalézt optimální test vedoucí k určení diagnózy. Tyto systémy jsou tvořeny rozhodovacími stromy sestavenými na míru k řešení problému v daném oboru, jde o **dopředné řetězení**. Naproti tomu **plánovací** expertní systém používá generátor možností, které jsou dále omezovány pouze na takové, které připadají v úvahu. Tyto možnosti jsou pak testovány podle hodnotících podmínek. Jde o takzvané **zpětné řetězení**. Kombinací Diagnostického a plánovacího expertního systému je nazývána **hybridní systém** [6].

Ať už diagnostické nebo plánovací, oba tyto systémy jsou založeny na pravidlech. Naproti tomu hybridní systémy jsou založeny na jiném principu. Základ hybridního systému může tvořit například: sémantická síť, rámce, objekty nebo systém typu tabule.

4.1 Části expertních systémů

Expertní systém se skládá z následujících modulů: báze znalostí, inferenční mechanismus, I/O rozhraní, vysvětlovací modul (pro zajištění potřebné efektivity nebude použit) a modul pro získávání znalostí (ten bude omezen pouze na načtení znalosti ze souboru). Prázdný expertní systém neobsahuje bázi znalostí a je teoreticky použitelný pro jakýkoliv problém. *Tento typ expertních systémů se podařilo vyvinout pouze pro řešení diagnostických úloh (diagnostické expertní systémy). Plánovací a hybridní expertní systémy mají totiž výrazně problémově závislou bázi znalostí* [13].

4.1.1 Báze znalostí

Báze znalostí obsahuje znalosti z dané oblasti. Obecně mohou být tyto znalosti nejrůznějšího charakteru: množina pravidel, matematické výrazy, rozhodovací stromy, sémantické sítě nebo objekty. Tyto znalosti zahrnují jak znalosti obecného charakteru, tak znalosti vysoce specifické pro daný účel. Součástí báze znalostí je i báze faktů obsahující data pro řešení daného problému. V případě karetní hry Taroky báze znalostí bude obsahovat aktuální stav hry a zakódovaná pravidla hry, aby expertní systém nezkoušel zahrát něco, co v dané situaci nelze.

4.1.2 Inferenční mechanizmus

Inferenční mechanizmus je soubor algoritmů, kterými lze na základě zadaných faktů z báze znalostí odvodit řešení problému. Tento mechanizmus je založen na pravidlech pro odvozování nových poznatků a způsobu prohledávání báze znalostí. Způsoby prohledávání báze znalostí jsou následující: dedukce (odvození závěrů plynoucích přímo z předpokladů), indukce (postup od specifického případu k obecnému), abdukce (usuzování směřující ze správného závěru k předpokladům, které jej mohly způsobit), heuristiky (pravidla „zdravého rozumu“ založená na zkušenostech), generování možností a jejich následná korekce (metoda pokus omyl), analogie (nalezneme nejpodobnější příklad a odvodíme závěr podobně jako v nalezeném příkladě), intuice (logicky téměř nevysvětlitelný způsob usuzování, snaha o napodobení lidského myšlení, bohužel zatím nebyl dobře implementován).

4.2 Systémy založené na pravidlech

Systémy založené na pravidlech se rozhodují na množině pravidel typu:
IF (předpoklady) THEN (následky) nebo
IF (předpoklady) THEN (následky1) ELSE (následky2)

Předpokladu neboli levé straně pravidla se říká *antecedent*. Je to obvykle množina tvrzení spojených logickou spojkou AND nebo OR. Naproti tomu následkům (pravá strana pravidla za THEN) se říká *konsekvent*. Tato množina tvrzení je na rozdíl od předpokladů spojena pouze logickým AND. Pozor, pravidla jsou občas špatně chápána jako implikace, což není správné. V expertních systémech, pokud je splněn předpoklad, tak se provedou pravé strany pravidla. *Naproti tomu implikace je definována pravdivostní tabulkou a její význam může být v přirozeném jazyce vyjádřen řadou způsobů* [14].

Tyto systémy se od běžných logických systémů vyznačují nemonotónním uvažováním a možností pracovat s neurčitou informací.

V zásadě existují dva přístupy vyhodnocování systémů založených na pravidlech: pomocí inference pravidel nebo pomocí klasifikace. Inference pravidel využívá *modus ponens*. Jsou-li splněny všechny předpoklady pravidla, jsou provedeny jeho následky. Jako následek muže být přidáno nové pravidlo nebo fakt. Případně naopak může být pravidlo nebo fakt zrušen. Toto řešení vede na inferenční sít. Klasifikací nejsou fakta navázána na pravidla již od začátku, ale navazují se až za běhu. Tyto systémy jsou obecnější a lze je používat i při plánování a syntéze. Hůř se zde implementuje podpora neurčitosti a u rozsáhlých aplikací hrozí snížení efektivity z důvodu časově náročného vyhledávání aplikovatelných pravidel[14].

4.3 Návrh expertních systémů

Pro návrh expertních systémů vznikl nový obor takzvaného *Znalostního inženýrství (knowledge engineering)*. Při návrhu expertního systému se setkávám se třemi hlavními problémy: způsob zápisu a reprezentace znalostí, efektivní způsob odvozování znalostí a zpracování neurčité informace.

Znalosti dělíme na deklarativní a procedurální. Podobně je tomu, programujeme-li v programovacím prostředí Prolog viz [15]. Deklarativní znalosti jsou elementární, například: „Jednička je celé číslo“. Naopak procedurální znalost je spíše pravidlem pro odvození, například:

„A a B lze sečíst, pokud A je celé číslo a současně B je celé číslo“. Téměř každou deklarativní znalost lze zapsat i procedurálně: „X je celé číslo, pokud X je jednička“. Znalosti expertních systému jsou většinou vnitřně reprezentovány speciální databází, která je optimalizovaná na rychlost vybavování. Často se pro tuto optimalizaci používá sémantického sdružování a sémantických sítí [16].

S neurčitostí se vypořádáváme většinou pomocí expertních systémů založených na Fuzzy interferenci nebo na teorii hrubých množin. Další možností, jak zpracovávat neurčitost, je přidat ke znalostem jejich pravdivostní hodnotu, například hodnotu v intervalu od 0 do 1. Toto vede na sestavení jednoduché inferenční sítě (detailní vysvětlení v [16]).

4.4 Požadavky na expertní systémy

V této kapitole bych chtěl prozkoumat výhody a nevýhody jednotlivých expertních systémů. Také bych chtěl rozhodnout, který expertní systém bude pro hraní karetní hry a pro implementaci na mobilním zařízení nejvhodnější. Z pohledu mobilního zařízení bych definoval požadavky na expertní systém následovně:

Paměťová náročnost běhu takovéhoto systému nesmí přesáhnout, paměťové možnosti mobilních zařízení respektive specifikace MIDP 2.0 (viz kapitola 3.2). A ne jen přesáhnou musí zbýt i nějaká paměťová rezerva na ostatní části aplikace. Uvědomme si, že se jedná o grafickou aplikaci.

Podobně bych omezil i náročnost na statickou paměť J2ME. Pro hru taroky bude báze znalostí uložena v Resource manageru (RMS), společně s případnými dalšími bázemi znalostí a limit pro tuto paměť je podle specifikace pouhých 8kB. Navíc se do této paměti musí vejít uložení stavu aplikace při přerušení, aby se z tohoto stavu přerušení aplikace byla schopna znova dostat do podobného stavu, jako byla před přerušením. Slovem „podobného“ myslím takového, aby uživatel nepoznal rozdíl a aby nemusel odehrát znova svých posledních pět tahů a tak podobně.

Co se týče nároků na výpočetní výkon, ty už nejsou tak kritické. Jde o komfort a hratelnost dané hry. Například bude-li každý hráč uvažovat zhruba dvě vteřiny nad svými možnostmi tahu, je tato situace pro mě akceptovatelná. Lidský hráč uvažuje nad tahem ve hře většinou mnohem delší dobu. Pokud by ale naopak čas uvažování hráče překročil dvacet vteřin, bude již dle mého názoru hra nehratelná, protože lidský hráč by tak mohl čekat i víc než jednu minutu na svůj další tah, což je i na mobilní hru docela dlouhá doba. Na druhou stranu, pokud by byl dán nějak časový limit pro uvažování, mohla by tak být logika počítačového hráče snadno rozdělena do několika úrovní pomocí jednoduchého časového limitu pro odpověď.

Z pohledu náročnosti dané hry je nezbytnou podmínkou schopnost daného systému modelovat a řešit situace vzniklé při hře. Nedokáže-li daný systém řešit plánovací úlohu, je sice možné jej použít, ale bude velice náročné vytvořit pravidla pro tento systém tak, aby se projevoval skutečně inteligentním chováním.

4.5 Jazyk expertního systému

Mezi požadavky patří možnost měnit umělou inteligenci bez překompilování celého programu, je tedy potřeba nalézt případně navrhnout jazyk expertního systému vhodný pro implementaci na mobilním telefonu. V současné době existuje mnoho jazyků pro zápis znalostí expertních systémů

jako třeba jazyk KIF (Knowledge interchange format) [17], Ontolingua (objektový jazyk odvozený z KIF) [18]. Dalším jazykem je jazyk OCML [16].

Můj expertní systém má naproti tomu jazyk o hodně jednodušší, vše je zakódováno do číselných hodnot a pravidla jsou v co nejjednodušší formě s ohledem na to, že celý expertní systém je implementován v J2ME a předpokládám užívání na mobilních telefonech, které většinou mají silně omezené paměťové a výpočetní prostředky (jak je popsáno v kapitole 3). Obecně pravidlo se skládá z podmínky a akce, která se má vykonat, pokud je podmínka splněna. Podmínka je ve své podstatě dotazem nad bází znalostí a akce umožňuje modifikovat bázi znalostí, tedy do ní přidávat nové pravidla nebo odstraňovat již existující pravidla.

4.6 Způsoby návrhu pravidel

Nyní jde o to, kde a jak získat množinu pravidel pro daný expertní systém. První možností, kterou se určitě pokusím použít, je zeptat se experta (hráče této hry), jaké pravidla zhruba používá při rozhodování v daných situacích. Podle čeho se rozhoduje, jaký typ hry bude dražit za jakých podmínek, případně jak se rozhoduje v případě hry samotné, kterou kartu vynese. Jak zhruba hodnotí vnitřně on jednotlivá kola. Podle toho, co jsem si všiml při pozorování profesionálních hráčů, často nevyužívají pro hodnocení kola hloupě jen hodnotu karet sebraných v daném kole. V některých případech je pro hráče výhodnější, například pokud jde o plané kolo (zde myšleno jako kolo s nízkou bodovou hodnotou), aby vzali daný zdvih protihráči, kteří tak vyplýtvají cenných trumfů a podobně. Takto získaná pravidla převedu do formy, které porozumí počítač. A porovnám jejich jednotlivé kombinace mezi sebou.

Další možností, pro získaní znalostí o vzájemném vztahu jednotlivých rozhodnutí a jejich užitků, je pomocí technik *Dolování dat z databáze Data mining*. Proto nutno sehnat nebo nějak vytvořit databázi odehraných her a v takto vytvořené databázi se pokusit nalézt další závislosti převoditelné na pravidla expertního systému. Tento způsob by mohl být účinný zejména ve způsobu rozhodování se při dražbě. Cílem toho bude zjistit, které atributy mají největší vliv na výsledek hry, například: poměr počtů taroků a ostatních karet nebo některé konkrétní kombinace karet. K účelu dolování bych chtěl použít ve škole dostupný systém SAS Data miner.

Pravidla a závěry získaná dolováním dat budou konzultována s hráči Taroků i testována jako celek proti starším verzím umělé inteligence. Pokud budou nové závěry správné a budou ve hrách vykazovat lepší úspěšnost než starší verze, budou použity. Takovéto testování bude sloužit nejen jako zpětná kontrola, ale také jako hodnocení úspěšnosti daných systémů pravidel. Pokud některý systém bude vykazovat statisticky výrazně lepší výsledky v hraní hry, budu automaticky předpokládat, že je lepší.

5 Aplikace Taroky pro mobilní zařízení

V rámci diplomové práce byla navržena a následně implementována aplikace pro mobilní zařízení umožňující hraní hry Taroky. Aplikace, jak již bylo řečeno v kapitole 3, je „Midlet“ a pracuje pod konfigurací CLDC určené pro ty nejjednodušší mobilní zařízení, jakými jsou právě mobilní telefony. Aplikace využívá profilu MIDP2.0, který je určen pro tvorbu her (viz Kapitola 3.2).

Základ aplikace tvoří třída `TarokyGame`, rozhraní `Player` a množina hráčů implementujících toto rozhraní. `TarokyGame` reprezentuje hru taroků. Tím je myšleno od rozdání přes dražbu hry, hlášení, samotné hraní a nakonec sčítání bodů za hru. U této třídy se předpokládá spuštění ve vlastním vlákně a tato třída pak interaguje se čtyřmi hráči podobně, jako je tomu při skutečném hraní hry Taroky. Nejprve jsou karty zamíchány. Následně jsou rozdány, každému hráči je oznámeno, které karty dostal. Následuje dražba, kdy jsou hráči postupně hrou dotazováni, jakou hru chtějí dražit. Pokud chtějí dražit nějakou vyšší hru, jsou všichni hráči informováni o tomto stavu. Při dražbě hry je samozřejmě implementováno rozdělení talónu patřičným způsobem podle typu hry a také možnost dražit Varšavu za podmínek specifikovaných v pravidlech (viz kapitola 2). Hlášení je prováděno obdobně, hráč je dotázán, zda chce hlásit některou ze *zavazujících hlášek* a z jeho karet jsou zjištěny povinně hlášené *prozrazující hlášky*. Poté jsou všechny hlášky daného hráče oznámeny všem hráčům. Po hlášení je zde možnost udělit flek na hru nebo na některou zavazující hlášku (Pagát, Valát). Po úvodní fázi probíhá hra samotná, kdy jsou hráči dotazováni, jakou kartu chtějí použít. Ostatním hráčům je vždy oznámeno, který hráč použil jakou kartu. Na konci hry je možno z třídy `TarokyGame` zjistit, jak hra dopadla.

Aplikace jako celek dále nabízí možnost přidat nebo odebrat některou inteligenci hráče definovanou pravidly expertního systému. Dále je umožněno při spuštění hry pojmenovat hráče. Expertní systém a herní grafické uživatelské rozhraní jsou v aplikaci chápány jako hráči hry Taroky a musí reagovat na všechny podněty vzniklé při hraní hry.

5.1 Hráč definovaný expertním systémem

Expertní systém, jak je popsáno v kapitole Chyba: zdroj odkazu nenalezen, je definován v externím souboru a lze tedy vytvořit nové způsoby chování počítače při uvažování. Jak bylo řečeno dříve, aplikace umožňuje načtení takovéto definice chování hráče. Načtením je zde myšleno převedení do vhodné formy a uložení do RMS. Za běhu aplikace je nutno mít v paměti kompletní stav báze znalostí, protože jej téměř všechna pravidla expertního systému potřebuji znát, aby určily, zda jsou proveditelná. Tudy cesta k úspoře využívání paměti pro běh nevede. Proto je celá inicializační báze znalostí uložená v RMS bez jakýchkoli optimalizací. Při inicializaci expertního hráče je báze znalostí načtena a je dále editována pravidly podle potřeby.

Na druhou stranu pravidla načtena všechna současně být nemusí, proto jsou v RMS uložené jako jednotlivé položky. Sekvenční přístup k pravidlům expertního systému je dostatečný, proto není problém vždy načíst pouze jedno pravidlo, zkontrolovat jeho proveditelnost a případně provést akci tohoto pravidla. Další provedená optimalizace pravidel vychází z předpokladu existence jejich velkého množství, z nichž aplikovatelných bude jen menšina, proto se v bázi znalostí ukládají podmínky a akce pravidla separovaně. Použitím této optimalizace se expertní systém značně zrychlil,

protože přestalo být nutné načítat pro ověření splnitelnosti podmínky celé pravidlo, ale stačí pouze načíst podmínku pravidla. Akce se z RMS načtou, pouze je-li podmínka pravidla vyhodnocena jako splnitelná.

5.2 Menu mobilní hry Taroky

Menu a další nastavení mimo hru samotnou jsou tvořeny pomocí implicitních komponent J2ME. Takto vytvořené menu sice nevypadají esteticky moc pěkně, ale na druhou stranu jsou zobrazeny na všech zařízeních správně. Dalo by se říct, že jsou zobrazeny podle systémového vzhledu daného zařízení.

Alternativou tohoto nepěkně vypadajícího menu by mohlo být menu mnou vykreslované do patřičné komponenty, podobně jako je tomu v případě samotného hraní. To by s sebou neslo nevýhody spojené s nutností řešit jednotlivé možnosti uživatelského vstupu (různé druhy klávesnic, různé velikosti písma a jiné odlišnosti jednotlivých zařízení). Další alternativou by mohlo být použití některé knihovny nabízející již vytvořené rozhraní na principu vykreslování do komponenty. Tyto knihovny zajišťují jednotný vzhled a funkcionalitu na všech zařízeních[1]. Takovouto knihovnou je i například **LWUIT** nebo **J2ME Polish** [19].

Menu umožňuje načíst pravidla expertního systému ze souboru definovaného v kapitole 6. Při načítání je zobrazena případná chyba ve struktuře tohoto souboru. Inteligence je pojmenována pro jednoduchost podle souboru, ze kterého byla načtena. Pokud se soubor jmenuje stejně jako některá již načtená inteligence, je původní inteligence přepsaná nově načtenou inteligenci z tohoto souboru. Načtené soubory pravidel zůstávají uloženy ve statické paměti aplikace ve vhodné přichystané formě. Načtenou inteligenci je pomocí menu aplikace možno smazat, například bude-li nutno šetřit paměť.

Druhou důležitou částí menu je možnost správy rozehraných her. Pokud je založena nová hra, tedy nadefinována jména a typy hráčů a případné startovní skóre, tak je tato hra uložena do vnitřní statické paměti aplikace. Po odehrání hry je v této hře změněno skóre jednotlivých hráčů dle výsledku hry. Takto vytvořené hry jsou reprezentovány v menu aplikace souborem jmen hráčů a je možné tyto hry mazat nebo pokračovat v jejich hraní. Majitel zařízení tak může mít rozehrané hry s více přáteli, se kterými se setkává v běžném životě (případně jejich různými kombinacemi). Zbytek protihráčů do počtu jim pak doplní vnitřní inteligence aplikace.

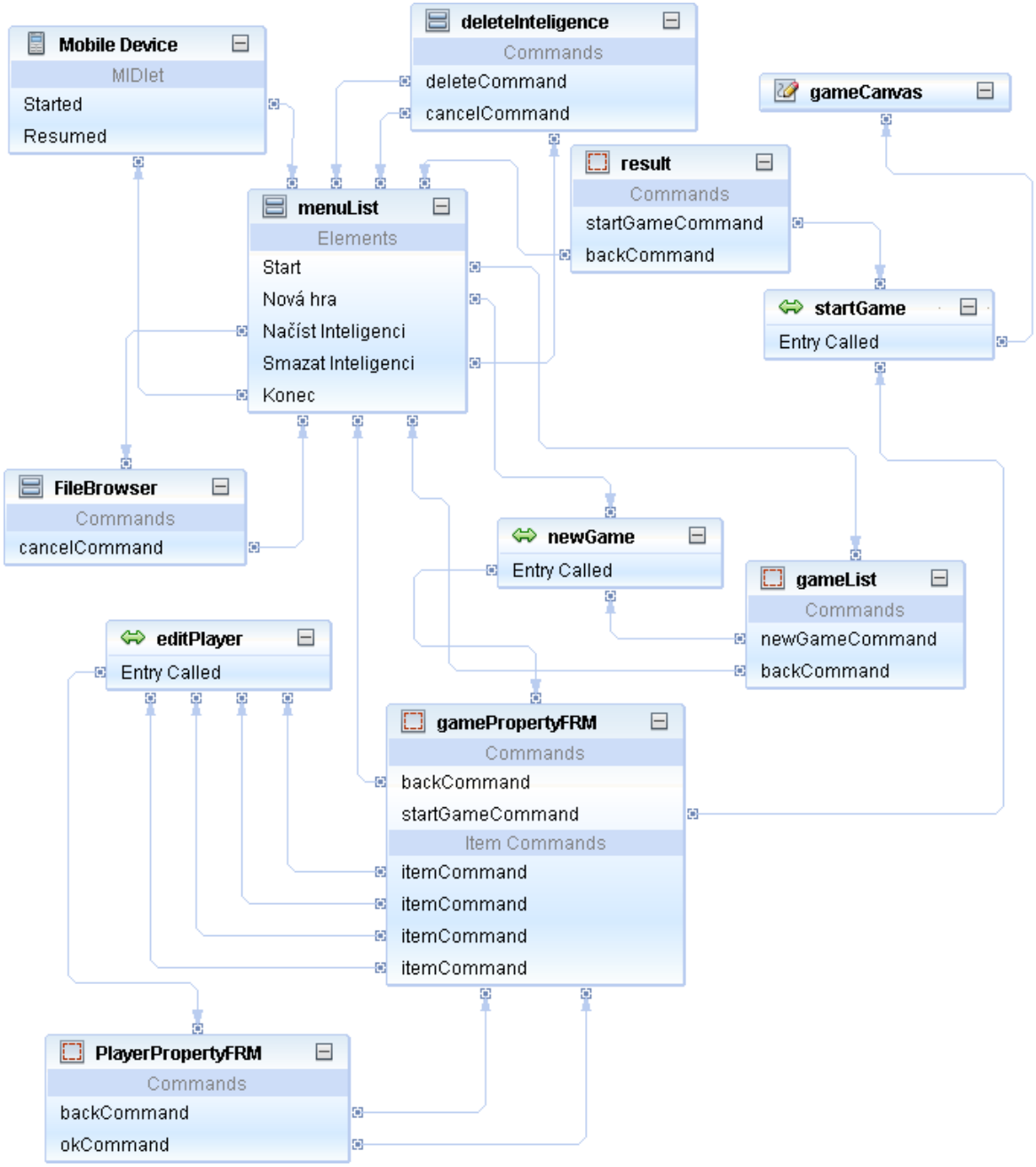
Nejlépe je funkčnost menu vidět na obrázku 4 vyjadřujícím, jak si jednotlivé komponenty menu mezi sebou předávají správu displaye. Vstupním bodem je *Mobile device*, který předá řízení hlavnímu menu při spuštění aplikace. V diagramu je vidět `gameCanvas`, což je komponenta starající se o hraní hry samotné (viz kapitola 5.3). Stručný popis dalších komponent na obrázku 4:

`deleteInteligence`	Vypíše jednotlivé již načtené inteligence a umožní jejich smazání.
`menuList`	Hlavní menu aplikace obsahuje položky vypsané v obrázku.
`fileBrowser`	Umožňuje procházení souborovým systémem zařízení, aby bylo možno vybrat soubor definující pravidla expertního systému.
`result`	Zobrazuje výsledek hry a je automaticky naplněn při dokončení hry.
`gameList`	Výpis rozehraných her které je možno smazat nebo spusit.
`PropertyFRM`	Formulář umožňující nadefinovat vlastnosti hry před jejím spuštěním.

1 https://lwuit.dev.java.net/

`PlayerPropertyFRM` Formulář pro editaci hráče tedy změny jména a typu hráče.

Ostatní jakoby komponenty jsou pouze různé pomocné spouštěcí procedury, které inicializují obsah následujících komponent. Případně tyto komponenty složitějším způsobem, který není z grafu zřetelný, přepínají. Například startGame krom toho, že předá správu displeje, spustí hru v paralelním vláknu. A postará se o zobrazení výsledku hry po jejím skončení.



obrázek 4: Flow diagram předávání správy displeje mezi jednotlivými komponentami menu vytvořeno při návrhu a programování aplikace v NetBeans IDE

5.3 Rozhraní pro hraní hry Taroky člověkem

Hraní hry, jak již bylo řečeno, je zobrazováno instancí třídy `GameCanvas`. Rozhraní pro hraní je pravým opakem menu, co se týče programátorského přístupu. Jedná se o grafickou aplikaci a nejsou zde používány žádné vnitřní vysokoúrovňové komponenty. Vše, co se při hraní zobrazuje, je vykreslováno na obrazovku a většina změn vyžaduje překreslení.

Ve spodní části obrazovky jsou vykresleny karty hrajícího člověka. Půlka karty je jakoby pod spodním okrajem displeje, to je možné, protože viditelná půlka přesně definuje, o jakou kartu se jedná. Toto je důležité zejména pro zobrazení na nejmenších mobilních obrazovkách, kdy je každý pixel místa podstatný. Karty soupeřů nejsou vidět a to ani jejich rubové strany, protože pro hru samotnou to není podstatné a také se tím ušetří prostor na obrazovce. Hráč jakoby sedí u stolu společně s ostatními hráči a na obrazovce se mu objevují hlášení a fleky od ostatních hráčů ve třech možných pozicích: uprostřed levého okraje od hráče po levici, ve středu pravého okraje od hráče po pravici a u horního okraje od hráče naproti danému hráči. Na středu hry se pak objevují hlášení hráče samotného. Pokud má hráč rozhodnou, jakou hru dražit nebo na co udělit flek a tak dále, je vykreslen dotaz a možné odpovědi. Hráč pak jednoduše vybere svou odpověď.

Každý lidský hráč je přímo reprezentován instancí třídy `Human`. Tato instance komunikuje s hrou Taroky a zobrazuje hrajícímu člověku výzvy a oznámení na hrací plochu. Pokud hraje více lidí, tak není dost možné, aby se například čtyři lidé současně dívali na jeden malinký displej mobilního telefonu. Hra je koncipována tak, aby si lidé telefon podávali a každou chvíli viděl obsah displeje pouze jeden člověk. Při střídání hráčů, a tedy i posunu telefonu dalšímu člověku, je na displeji zobrazena čekací obrazovka. Čekací obrazovka upozorňuje, na to kdo je dále na tahu a komu je tedy třeba předat telefon. Podobně je tomu i například v některých počítačových hrách typu tahová strategie, které umožňuji hru více lidí na jednom počítači.

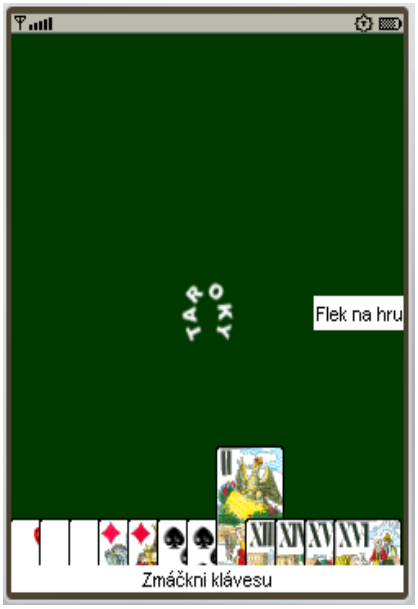
Přepínání hráčů je navrženo a implementováno s ohledem na minimalizaci počtu předání telefonu. Z tohoto důvodu každý hráč má jakousi frontu událostí, které nepotřebují vstup od člověka. Události v této frontě jsou provedeny, jakmile je hráč na tahu. Hráč se na tah dostává, když je nutno rozhodnout se, co v dané situaci zahrát (co hlásit, nebo jakou kartu shodit). Dále je nutno zajistit, aby tento systém neprozradil žádnou informaci o stavu karet hráče. Proto se hráč musí dostat k telefonu, i pokud je z pohledu ostatních hráčů možno v jeho situaci něco hlásit, i když z obsahu karet hráče nevyplývá žádná možnost rozhodnutí. To nastává, pokud jsou hlášeny zavazující hlášení. Hráč, který by se nedostal na tah, by byl prozrazen, že nemá ani *Pagáta* a ani *Škýze*, aby mohl hlásit hlášky *Pagát* nebo *Valát*. Podobná situace je při udělování fleků. Pokud by byl některý hráč přeskočen, došlo by ostatním, že nemohl udělit flek. Z pravidel vyplývá, že nikdo nesmí udělit flek na cokoli spoluhráči, pokud je si vědom skutečnosti, že hrají spolu.

Při hraní hry je povoleno hlásit i zdánlivě nesmyslné hlášení například *Valát* proti třetí povinnosti. Je to divné, ale žádná pravidla hry to přímo nezakazují, a tak nebylo, podle čeho se řídit. Na druhou stranu hlásit *Valát* proti nevhodně dražené třetí povinnosti je výhodné.

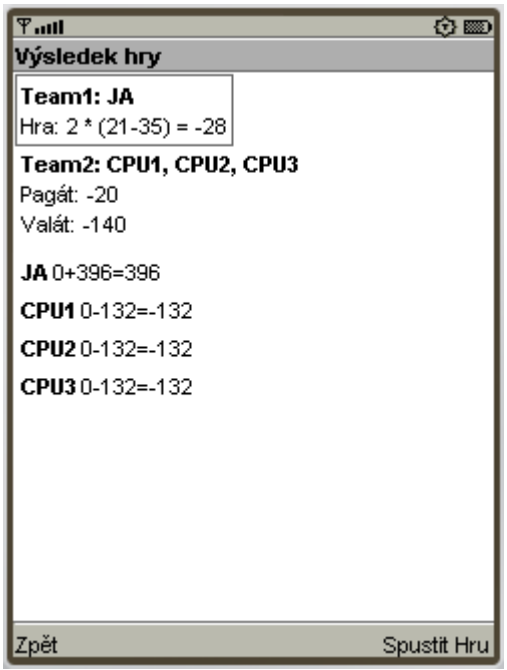
obrázek 5: Nastavení nové hry

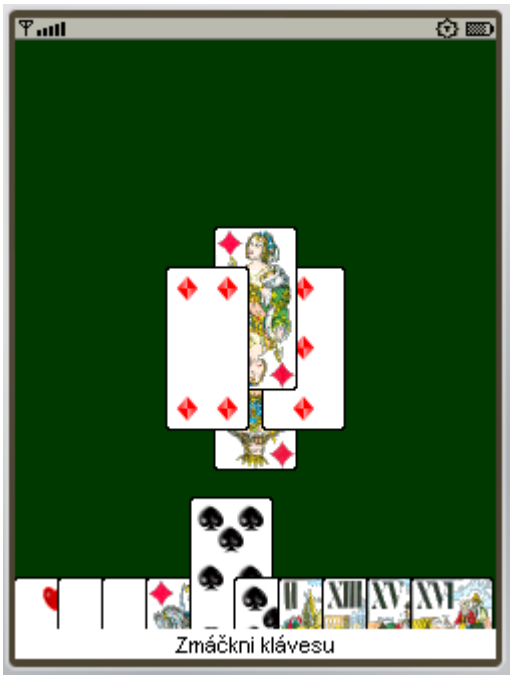
obrázek 6: Dražba typu hry

obrázek 7: nabídka karet - hra Trojka



obrázek 8: Hráč po mé levici flekuje hru





obrázek 9: Samotná hra

obrázek 10: Zobrazení výsledků hry

6 Jazyk popisující inteligenci hráče Taroků

Pro popis inteligence jsem navrhl jednoduchý jazyk vycházející nejen z poznatků a požadavků na jazyk expertního systému, ale také z omezení možností mobilních zařízení. V následujících podkapitolách se pokusím nadefinovat, jak syntaxi tak sémantiku tohoto jednoduchého jazyku.

Inteligence pro hraní hry taroky je definována zdrojovým souborem obsahujícím jeden povinný blok (`PROLOG`) a deset volitelných bloků (`SETCARD`, `SETBIT`, `GETBIT`, `SETBONUS`, `GETBONUS`, `SETTURN`, `GETTURN`, `SETTAROKDISCART`, `SETFLEK`, `GETFLEK`). Volitelný blok je definován svým názvem a množinou pravidel uzavřenou do složených závorek. Každý z deseti volitelných bloků definuje chování hráče definovaného tímto expertním systémem v určité fázi hry.

ZDROJ_SOUBOR ::= ***PROLOG*** *'{' {PROLOGTERM} '}' {****SETCARD*** *'{' {Rule} '}'} {****SETBIT*** *'{' {Rule} '}'} { ****GETBIT*** *'{' {Rule} '}'} {****SETBONUS*** *'{' {Rule} '}'} { ****GETBONUS*** *'{' {Rule} '}'} { ****SETTURN*** *'{' {Rule} '}'} {****GETTURN*** *'{' {Rule} '}'} {****SETTAROKDISCART*** *'{' {Rule} '}'} {****SETFLEK*** *'{' {Rule} '}'} {****GETFLEK*** *'{' {Rule} '}'}*

PROLOGTERM ::= "jednoduchý výraz, zpracovatelný knihovnou mProlog, ukončený znakem '.'"

V celém souboru je navíc možnost psát řádkové komentáře kdekoli, kde je povoleno umístit nový řádek. Za komentář jsou považovány všechny znaky od úvodních dvou lomítek „//“ až po konec řádku. Pokud některý z volitelných bloků neexistuje nebo je prázdný, zvolí expertní systém náhodně jedno z možných řešení dané situace (například vybere náhodnou hratelnou kartu). Minimální soubor popisující expertní systém tedy obsahuje prázdný povinný blok PROLOG. Rozhodování definované takovým souborem je náhodné. Náhodné chování odpovídá úrovni začátečníka, který pochopil pravidla hry, ale zatím nechápe hlubší expertní souvislosti a nerozvinul si žádné strategie pro hraní této hry.

6.1 Povinný blok PROLOG

Tento blok je vždy v souboru definující rozhodování hráče uveden jako první. Definuje obsah inicializační báze znalostí expertního systému. Programátor zde, jak doufám, umístí počáteční nastavení (fakta platící na začátku každé hry) hráče pro hru a předdefinuje si odvozovací pravidla (klauzule). Báze znalostí je reprezentovaná prologem, jak již napovídá název tohoto povinného bloku. Báze obsahuje data v takovém tvaru, aby bylo možno je zpracovávat interpretem mProlog. K otestování možností knihovny mProlog slouží aplikace MProlog test (viz 6.1.2). Jednotlivé klauzule jsou podobně jako v jazyce Prolog odděleny znakem „.“ (tečka). Pro zjednodušení načítání klauzule prologu nesmějí obsahovat znak tečku na jiném místě.

6.1.1 MProlog

Mnou navržený a implementovaný expertní systém využívá k ukládání znalostí, odvozovacích pravidel a odvozování *mProlog*. Jedná se o odlehčenou implementaci interpretu prologu určenou pro běh na mobilních telefonech. Oproti specifikaci prologu se mProlog ve svém chování v mnohém liší. Následující rozdíly jsem objevil při jeho užívání a při zkoumání návodu na jeho užívaní [20].

Prvním a asi nejhorším problémem je operátor „|“, který v implementaci mPrologu zcela chybí. Tento operátor slouží v prologu pro práci s obecně dlouhými seznamy. Pomocí tohoto operátoru lze běžně rozdělit libovolně dlouhý seznam na takzvanou hlavičku či více hlaviček a tělíčko. Příkladem běžného užití tohoto operátoru budiž zápis `[X,Y|Z]`. Pokud tímto zápisem unifikujeme seznam `[1,2,3,4,5]`, pak jsou proměnné unifikovány takto: `X=1`, `Y=2` a `Z=[3,4,5]`. Toto je asi jediný způsob, kterým v prologu implementovat průchod seznamem, tedy pomocí rekurze. V implementaci mPrologu jsem nic podobného nenašel, pouze možnost přidat prvek na konec seznamu pomocí operátoru „+“. Bohužel podobně jako přes ostatní operátory neprojde unifikace, tudíž této konstrukce nelze využít k procházení seznamu. Neexistence tohoto operátoru způsobuje problémy, se kterými je třeba počítat při návrhu pravidel a struktury báze znalostí pro jednotlivé definice umělé inteligence. MProlog celkově špatně pracuje se seznamy například: `test([A,3]):-eq(A,4)` je při volaní `test(X)` výsledek `X->[A,3]`, pokud voláme `test([X,Y])` je unifikováno správně `X->4`, `Y->3`.

Druhým dost podstatným rozdílem oproti specifikaci prologu jsou předdefinované predikáty. MProlog obsahuje předem definované predikáty, jejichž význam a tvar se od standardní verze liší. Tato odlišnost je způsobena hlavně prostředím, pro které je daná verze prologu určena. Následuje výčet předdefinovaných predikátů:

`eq(X,X).`	ekvivalence obousměrná unifikace volné proměnné na vázanou je možno takto provádět vyčíslení hodnoty v prologu běžně prováděné pomocí klíčového slova „is“ (`ex(X,1+1). X->2`)
`and(X,Y).`	logický součin (pracuje s hodnotou `true`)
`or(X,Y).`	logický součet (podobně jako `and(X,Y)` s hodnotou `true`)
`assert(X).`	přidá do báze znalostí predikát `X`
`retract(X).`	odebere s báze znalostí predikát `X`
`random(MAX, X).`	unifikuje `X` na náhodnou hodnotu z intervalu <0, MAX)
`random(MIN, MAX, X).`	unifikuje `X` na náhodnou hodnotu z intervalu <MIN, MAX)
`write(X).`	vypíše `X` vhodné použít pro pomocné výpisy při ladění pravidel expertního systému
`nl.`	odřádkuje výstup
`write(Stream, X).`	vypíše `X` do Javou otevřeného streamu (netestováno, pro potřebu mobilní hry nepoužitelné)
`if(X,Y,Z).`	pokud je `X` pravdivé, vykoná se `Y` jinak `Z`

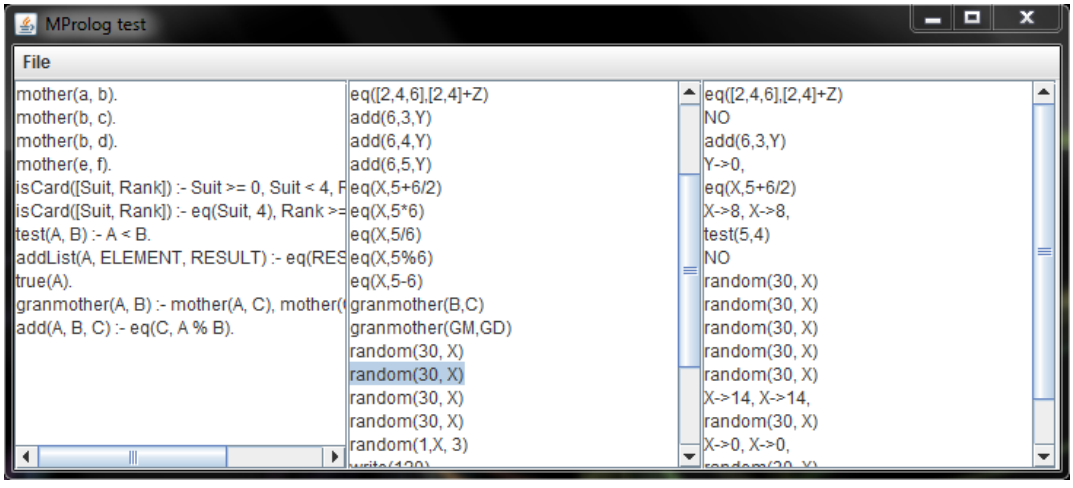
Třetím rozdílem je výsledná unifikace proměnných. MProlog nepřekládá termy pro proměnné do vnitřní unikátní reprezentace, jako to dělá například SWI-Prolog, který každou proměnnou reprezentuje jako _G+unikátní_číslo. Z tohoto důvodu při výsledné unifikaci mProlog nalezne i unifikace všech proměnných se stejným názvem jako mají proměnné dotazu. Při tomto nedochází k chybě výpočtu, pouze je nejednoznačné, který výsledek je hledaným. Například pokud dotaz používá proměnnou `XXX` a některé pravidlo také proměnnou `XXX`, pak výsledná unifikace `XXX` je

dvojitá jednou pro proměnnou z pravidla a podruhé pro proměnnou z dotazu. Toto nedělá problémy, pokud si je toho programátor vědom. Stačí oddělit jména proměnných v bázi znalostí od jmen proměnných v dotazech, například nějakým prefixem nebo jiným podobným vhodně zvoleným dělením jmenného prostoru. Při experimentování s knihovnou mProlog byly proměnné vždy vráceny v takovém pořadí, kdy hledaná proměnná byla jako poslední. Rozhodl jsem se tedy proměnné ukládat do hash tabulky. Pokud do hash tabulky vložím další hodnotu pod stejným klíčem, provede se přepsání původní hodnoty. Tím je zajištěno, aby byla výsledná unifikace proměnné správná, pokud bude platit předpoklad o pořadí vrácených proměnných. Problém nastává, pokud správná unifikace proměnné neexistuje, ale při odvozování platnosti byla unifikována proměnná se shodným jménem. V takovém případě je jako výsledná unifikace vrácena unifikace této nežádoucí proměnné. Tato definice je kostrbatá a nepochopitelná, proto se tento problém ještě jednou pokusím vysvětlit na příkladu. V bázi znalosti je predikát `chyba(XXX,Y)` a dotaz na bázi znalostí je položen takto `chyba(23,XXX)`, potom výsledek dotazu bude samozřejmě správně pravda, ale výsledná unifikace `XXX` bude 23 a to je samozřejmě špatně. Je nutné, na to při psaní pravidel a báze znalostí myslet a striktně oddělit jmenné prostory, aby k podobným případům nemohlo dojít.

Pro testování možnosti knihovny mProlog jsem vytvořil jednoduchou aplikaci nazvanou **MProlog test**.

6.1.2 Aplikace MProlog test

Tato desktop aplikace umožňuje testovat možnosti knihovny mProlog. Protože mProlog nevyužívá žádné specifické prostředky z J2ME, které by v Java Standard Edition (J2SE) nebyly možné použít, je tato testovací aplikace vytvořena v J2SE. Myslím si, že bude nutná pro návrh pravidel a struktury báze znalostí jednotlivých hráčů. Nejedná se ani tak moc o celou aplikaci jako spíš o uživatelské rozhraní pro knihovnu. Okno aplikace se skládá ze třech výrazných částí (viz obrázek 11). Úplně vlevo je textová reprezentace báze znalostí. V prostřední části je možnost napsat dotaz nad bází znalostí. Vpravo je pak vidět, jak dotaz dopadl. Je zde přesměrován i standardní výstup, takže lze testovat i funkce pro psaní na výstup.



obrázek 11: Rozhraní aplikace MProlog test

Dotaz na bázi znalostí provedeme označením text dotazu (v prostřední části okna) a kliknutím na označené kolečkem myši. V pravé části okna se ihned zobrazí výsledek dotazu jako opsaný text dotazu následovaný řádky, které reprezentují jednotlivé možnou unifikaci proměnných dotazu. Pokud dotaz neunifikuje žádné proměnné, zobrazí se výsledek ve tvaru YES nebo NO podle toho, jestli lze nebo nelze z báze znalostí odvodit. V současnosti je také aktualizována báze znalosti, aby bylo možno testovat i funkce assert a retract (viz kapitola 6.1.1).

6.2 Ostatní volitelné bloky

Zde je popsán význam jednotlivých volitelných bloků pravidel. Jsou zde přesně definovány jejich vstupy a očekávané výstupy. V zásadě lze bloky rozdělit do dvou skupin: bloky typu „set“ a bloky typu „get“. Chování bloků v rámci jedné skupiny je podobné, což snad pomůže programátorovi v orientaci v kódu bez nutnosti pamatovat si veškeré detaily. Volitelné bloky odpovídají jednotlivým funkcím definovaným v rozhraní hráče pro hru taroky (interface `Player`).

Set bloky slouží k informování expertního systému o nějaké události ve hře. Nahrazují jakýsi pohled skutečného hráče na dění ve hře. Také by se dalo říct, že se jedná o vstupní bloky. Například vidí-li skutečný hráč, jak nějaký jeho soupeř shodil kartu, je expertní systém informován o této skutečnosti provedením příslušného volitelného bloku. Expertní systém má tak k dispozici všechny relevantní informace o stavu hry. Předpokládaná funkce set bloků je vytváření modelu hry ve vhodné formě pro dotazy. Může zde ovšem být i složitější logika již pro odvození některých znalostí, zejména pak je-li nutno šetřit paměť pro popis modelu hry. Základní expertní systém by mohl například pouze počítat počty taroků, které už šly, a na základě toho rozhodovat, jakou kartu v jaké situaci použít. Mojí původní snahou bylo reprezentaci hry navrhnout bez použití tohoto typu bloku. Tyto vstupy by pak byly vkládány přímo do báze znalostí expertního systému. Nenašel jsem vhodnou reprezentaci dat, kterou by bylo možno efektivně provádět všechny případné expertní pravidla.

Naproti tomu bloky typu „get“ většinou nemají žádné vstupy a expertní systém očekává po jejich provedení v bázi znalostí výsledek reprezentující rozhodnutí dané situace. Tyto bloky jsou klíčové a předpokládám, že v nich bude zakódovaná velká část inteligence daného hráče.

Tyto bloky obsahují libovolné množství pravidel expertního systému. Prováděním bloku je myšlena kontrola, zda dané pravidlo je proveditelné a jeho případné provedení. Toto je provedeno pro všechna pravidla a tím je blok považován za vykonaný. Při provádění pravidla je hlavním cílem upravit bázi znalostí. K čemuž je možno využít unifikované proměnné z podmínky daného pravidla.

6.2.1 Blok SETCARD

Tento blok je proveden, pokud hráč dostane kartu. Dostat kartu hráč může ve dvou případech: při úvodním rozdávání nebo při rozebírání talonu. Tento blok je proveden pro každou kartu zvlášť. Tedy v případě rozdávání je blok proveden dvanáctkrát. Při provádění bloku je nastavena proměnná `CARD` reprezentující přijatou kartu jako dvouprvkový seznam `[barva, hodnota]`. `Barva` určuje barvu karty a odpovídá jedné z následujících konstant: 0 – herce (srdce), 1 – káry (koule), 2 – piky, 3 – kříže, 4 – taroky. `Hodnota` je index (počítáno od 0) karty v dané barvě podle pořadí, jak se karty přebíjejí. Příklad: Pagát bude předán jako `[4, 0]`, Škýz předán jako `[4, 21]` a dáma piková bude `[2, 6]`. Pro zpřehlednění lze do bloku PROLOG přidat axiomy: `pika(2)`, `tarok(4)`. Dotazem na

tyto axiomy pak určovat barvu. Ovšem povede to k jisté degradaci efektivnosti, což může být problém, zejména pokud se složitost pravidel expertního systému bude blížit k limitům možností daného typu zařízení.

6.2.2 Blok SETBIT

Tento blok je proveden, aby oznámil, co který hráč dražil za hru: bud nic, nebo vyšší typ hry. Také se pomocí tohoto bloku upřesňuje typ hry (s kým chci hrát první respektive druhou povinnost). Poslední varianta využití tohoto bloku slouží k oznámení, kolikáté karty bere hráč při třetí povinnosti. Při všech variantách jsou nastaveny proměnné `PLAYER` a `BIT`. Hodnota `PLAYER` je pozice hráče volícího hru. Volený typ hry je uložen v proměnné `BIT`. Pozice hráčů jsou vztaženy vzhledem k pozici expertního hráče vykonávajícího blok SETBIT (dále aktuální hráč): 0 – aktuální hráč, 1 – hráč po levici od aktuálního, 2 – hráč naproti aktuálního a 3 – hráč po pravici od aktuálního. Kódy jednotlivých druhů her (význam proměnné `BIT`) jsou uvedeny v následujícím přehledu:

-1	nic pokud hráč nehlásí nic	7	druhá povinnost s 19
0	Varšava	8	druhá povinnost s 18
1	první povinnost	9	druhá povinnost s 17
2	první povinnost s 19	10	druhá povinnost s 16
3	první povinnost s 18	11	trojka (preferanc)
4	první povinnost s 17	12	trojka s druhými kartami.
5	první povinnost s 16	13	trojka s třetími kartami.
6	druhá povinnost	14	sólo

Pokud je hlášena trojka, tak je hráč navíc informován, které karty jsou vidět z talonu. Jsou to ty karty, které by hráč viděl i ve skutečnosti. Pokud hráč dražíl trojku, tak tento hráč vidí první tři karty z talonu a ostatní nevidí nic. Pokud hlásí trojku s druhými kartami, tak vidí druhé tři karty a ostatní hráči vidí prvé tři, a pokud hraje trojku s třetími kartami, tak hráč vidí první tři karty a ostatní hráči vidí druhé tři karty z talonu. Tyto viditelné karty jsou zakódovány v proměnných `CARD1`, `CARD2` a `CARD3` do dvouprvkových seznamu stejně jako proměnná `CARD` v bloku SETCARD.

6.2.3 Blok GETBIT

Je vykonáván, pokud je potřeba, aby expertní systém rozhodl, jakou hru chce dražit. Při provádění tohoto bloku není zde definována žádná vnitřní proměnná. Expertní systém po skončení bloku GETBIT očekává v bázi znalostí klauzuli ve tvaru `return(X)`, kde `X` bude kód hry, kterou chce aktuální hráč hrát. Pokud je nalezeno více možných unifikací `X`, je vybraná náhodná z nich. Vrácený kód hry je dále samozřejmě kontrolován hrou, aby nebylo možno hlásit nesprávný typ hry pro danou situaci. Hráč je informován i o tom, co sám dražil pomocí bloku SETBIT.

6.2.4 Blok SETBONUS

Pomocí vykonání tohoto bloku pravidel se hráč dozví o hláškách ostatních hráčů. Při provádění jsou nastaveny proměnné `PLAYER` a `BONUS`. `PLAYER` označuje pozici hráče hlásícího nějakou hlášku podobně jako v bloku SETBIT. Proměnná `BONUS` pak obsahuje kód hlášky. Hlášek je celkem devět, z toho dvě zavazující (viz kapitola 2.1). Kódy hlášek jsou uvedeny v následujícím přehledu:

0	Taroky	5	Trul
1	Pagát	6	Honery
2	Valát	7	Barvy
3	Taročky	8	Královské honery
4	Barvičky		

V tomto bloku se předpokládá uložení závislostí vyplývajících z prozrazujících hlášení. Toto je podle mě zásadní bod v budování modelu hry. Hráči jsou zde povinni prozradit informace o svých kartách. Dokonce i hráč, který nic nehlásí, tak v podstatě vyjadřuje, že nemá ani dost taroků na taročky, ani dost barev na barvičky. Tyto závislosti a informace jsou velice důležité při následujícím rozhodování.

6.2.5 Blok GETBONUS

Opak k bloku SETBONUS. Tento blok je vykonáván, aby dal možnost hráči hlásit některou zavazující hlášku (Pagát nebo Valát). Po vykonání bloku je v bázi znalostí očekáván fakt `return(X)`, kde `X` je unifikováno na kód Valáta nebo Pagáta případně je tento fakt v bázi znalostí dvakrát, pokud chce hráč hlásit obě zavazující hlášení. Stejně jako v případě ostatních bloků očekávajících výsledek, jsou po skončení bloků z báze znalostí odebrány všechny fakta `return(X)`.

6.2.6 Blok SETTURN

Blok se vykonává, pokud některý hráč hrál kartu. Při vykonávání jsou přednastavené proměnné `CARD` a `PLAYER`. `CARD` obsahuje dvouprvkový seznam reprezentující hranou kartu. `PLAYER` obsahuje kód pozice hráče vzhledem k aktuálnímu hráči. Hráč reprezentovaný expertním systémem je tímto způsobem informován o tazích soupeřů. Předpokládá se udržování kontextu o stavu odhozených karet, aby se hráč mohl, až se dostane na tah, rozhodnout, kterou kartu bude hrát.

Pozor v případě typu hry Varšava je pomocí SETTURN oznámeno také shození páté karty. V takovém případě je hodnota proměnné PLAYER je rovna čtyři, což neodpovídá pozici žádného hráče.

6.2.7 Blok GETTURN

Tento blok je bez přednastavených proměnných vykonáván, pokud je hráč žádán, aby odhodil kartu. Odhození karty může být potřeba ze dvou důvodů: buď je hráč na tahu a je potřeba hrát, nebo hráč obdržel karty z talonu a je potřeba odložit patřičný počet karet (viz kapitola 2.1). Po skončení tohoto bloku se očekávaná v bázi znalosti fakt `return(X)`. Zde `X` odpovídá dvouprvkovému seznamu reprezentujícímu kartu z ruky hráče. Pokud je těchto faktů nalezeno více, je vybrán náhodně jeden z nich a ten je použit. Pokud se Expertní systém rozhodne hrát kartu, kterou nemá nebo v dané situaci hrát nemůže, je zahrána náhodná z hratelných karet.

6.2.8 Blok SETTAROKDISCART

Pravidla tohoto volitelného bloku jsou vykonávána jen výjimečně, pokud je některý hráč v úvodu hry při rozdělování talónu nucen odložit do balíčku sebraných karet některý tarok. Dle pravidel tato situace může nastat, pokud již na ruce nemá jinou kartu, která by se dala odložit. V takovém případě

se v normální karetní hře karta odkládá lícem vzhůru. V normální hře tedy všichni hráči vědí, kdo a jaký tarok odložil. Před vykonáváním tohoto bloku je nastavena hodnota proměnné `PLAYER` na pozici hráče, který odkládá tarok a hodnota `\CARD` dvouprvkový seznam kódující odkládaný tarok stejně jak je popsáno již dříve.

6.2.9 Blok SETFLEK

Tento blok je vykonáván, pokud některý hráč udělil flek na hru nebo Pagáta nebo Valáta. Před vykonáváním tohoto bloku je nastavena hodnota proměnné `PLAYER` na pozici hráče udělujícího flek. Dále je nastavena hodnota proměnné `FLEK` na kód toho, na co je flek udělen. Přehled možných hodnot proměnné `FLEK`: 1 – Pagát, 2 – Valát, 3 – hra. Jako pomůcka je zde také nastavena proměnná `FLEKCOUNT`, která říká, kolikátý flek byl udělen na danou záležitost. Pokud je tedy `FLEKCOUNT` nastaveno na jedna, je udělen flek, pokud na dvojku, je udělen kontra flek a pokud na trojku, je udělen super flek.

6.2.10 Blok GETFLEK

Tento blok se volá, pokud hra nabízí hráči možnost udělit flek na Pagáta, na Valáta nebo na hru. Bylo by divné, kdyby například hráč hrající proti třetí povinnosti mohl udělit flek na Pagáta jinému hráči než vydražiteli hry. Jinak by totiž uděloval flek vědomě vlastnímu týmu. Naopak existují situace, kdy je možno udělit flek spoluhráči, pokud z karet hráče udělujícího flek nevyplývá jednoznačně, zda jej uděluje spoluhráči nebo protihráči. Samozřejmě nelze udělit flek na to, co nebylo hlášeno. Aby expertní systém nemusel složitě zjišťovat, na co lze flek udělit v dané situaci, jsou před vykonáváním bloku nastaveny proměnné `PAGAT`, `VALAT` a `GAME`. Možné hodnoty těchto proměnných jsou 1, pokud je možno udělit flek a 0 pokud tato možnost není. Systém očekává `return(XXX)` kde `XXX` je rovno `1` pokud se má dát flek na Pagata, `2` pro flek na Valat a `3` pro flek na hru.

6.3 Pravidlo Expertního systému

Pravidlo expertního systému je podmínka syntakticky velice podobná podmínce v jazyce C nebo v jazyce Java. Sémantika pravidla má lehkou odlišnosti od běžné sémantiky podmínky. Podmínka totiž chystá proměnné pro akce pravidla. Akce pravidla pak s pomocí těchto proměnných upravují bázi znalostí.

```
RULE ::= IF '(' CONDITION ')' '{' {ACTION} '}'

CONDITION ::= MPROLOGTERM_EXTEND

CONDITION ::= '(' CONDITION ')'

CONDITION ::= NOT CONDITION

CONDITION ::= CONDITION AND CONDITION

CONDITION ::= CONDITION OR CONDITION

ACTION ::= add  MPROLOGTERM_EXTEND
```

*ACTION ::= **del** MPROLOGTERM_EXTEND*

*ACTION ::= **query** MPROLOGTERM_EXTEND*

MPROLOGTERM_EXTEND ::= MPROLOGTERM rozšířený o proměnné.

Rozšířením o proměnné zde myslím možnost použít v `MPROLOGTERM` řetězci proměnné uvozené znakem zpětného lomítka '\'. Tyto proměnné před vykonáváním textově nahrazeny jejich hodnotou. Pokud proměnná neexistuje, tedy nemá hodnotu, je textově nahrazena jménem. Například z `write(\XXX)` je nahrazeno `write(XXX)`, pokud proměnná `XXX` neexistuje.

Jak je vidět, každé pravidlo obsahuje dvě hlavní části podmínku a akce. Při ověřování zda podmínka je splnitelná, jsou vždy vyhodnoceny všechny formule. Pokud byly unifikovány nějaké proměnné na hodnoty, jsou přidány do hash tabulky proměnných. Takto vzniklé proměnné je možno použít přes zpětné lomítko podobně jako přednastavené proměnné. Každé pravidlo tedy má na začátku přednastavené proměnné podle bloku, ve kterém se vyskytuje a pokud jsou vykonávány akce pravidla, tak se k těmto implicitně nastaveným proměnným přidají všechny unifikace proměnných z podmínky tohoto pravidla.

Samotné akce pravidla jsou trojího druhu `add`, `del` a `query`. Akce `add` přidá následující pravidlo nebo fakt do báze znalostí. Akce `del` odebere následující pravidlo nebo fakt z báze znalostí a lze jej používat i v obecném tvaru například `del matka(X, Z).` V tomto případě odebere první nalezenou definici například `matka(petr,lucie).` Pozor akce `del matka(_,_)` neodebere nic protože s „_“ nelze nic unifikovat. Akce `del` také vytváří proměnné podobně, jako to dělá akce `query` nebo podmínka pravidla, díky tomu, je možno zjistit co expertní systém z báze znalostí odstranil. Více o způsobu provádění pravidel se lze dozvědět z následujících příkladů. Posledním typem příkazu je příkaz `query`, který slouží k dalšímu dotazování na bázi znalostí. Samozřejmě unifikované proměnné z takovéhoto dotazu jsou přidány do hash tabulky proměnných. Na první pohled se zdá, že se jedná o redundantní pravidlo, protože všechny tyto dotazy mohly byt pomocí `and` přidány do podmínky pravidla. Takovéto přidávání dotazu do podmínky pravidla by již tak složité podmínky dělalo ještě nepochopitelnější.

Příklad1: inkrementace počtu karet

Báze znalostí před prováděním pravidla	**Pravidlo**	**Stav báze znalostí po provedení pravidla**

pocetkaret(3).	`If( pocetkaret(POCET). ){`
	`del pocetkaret(\POCET).`
	`add pocetkaret(\POCET+1).`
	`}`
		pocetkaret(4)

Příklad2: vypsání na standardní výstup, kdo shodil jakou kartu. Pravidlo je umístěno v bloku SETCARD

Pravidlo

`If( write(hrac\PLAYER shodil kartu \CARD). ){`
`query nl.`
`}`

Příklad3: Chyba omylem přepsání hodnoty `\PLAYER` opět v bloku SETCARD voláno z hodnotu `PLAYER=3 CARD=[0,0]`

Báze znalostí před prováděním pravidla	**Pravidla**	**Stav báze znalostí po provedení pravidla**

naproti(2).	`If( NOT naproti(PLAYER). OR not false. ){`
	`add hodnotaplayer(\PLAYER).`
	`}`
		naproti(2).
		Hodnotaplayer(2).

Protože v podmínce je `PLAYER` místo `\PLAYER`, tak je unifikováno na hodnotu dva z báze znalostí a tím je proměnná `PLAYER` přepsána a v těle podmínky je poté použita tato nová hodnota proměnné.

7 Návrh pravidel expertního systému pro hraní hry taroky

Pravidla expertního systému jsem začal tvořit od těch nejjednodušších. Pokud byl přidán celek pravidel, která by teoreticky měly vést ke zlepšení úspěšnosti při hraní, byl proveden test porovnáním nově vzniklého souboru pravidel s ostatními. Pokud byl nový systém pravidel úspěšnější než dříve vytvořené soubory pravidel, byl k nim tento soubor přidán. Postupně tak vznikala stále lepší a lepší inteligence pro hraní hry.

Test úspěšnosti souboru pravidel jsem prováděl tak, že jsem nechal daný soubor pravidel hrát proti ostatním hráčům a sledoval jsem skóre po sto odehraných hrách případně po více, pokud rozdíl nebyl tak zřetelný, jak jsem čekal. Pro toto testování byla hra upravena tak, aby vždy čtyři po sobě následující hry byly rozdány s identickými kartami. Dále byl upraven expertní systém pro odstranění náhodnosti v případě, kdy z pravidel není odvozeno řešení. Při testování tímto způsobem tedy čtyři stejní hráči, kteří nemají náhodnost zavedenou přímo v pravidlech, po odehrání libovolného počtu kol dělitelného čtyřmi, dosáhnou původního skóre. Tedy pustím-li test stokrát pro čtyři hráče definované minimálním přijatelným souborem pravidel nazvaným Empty (viz kapitola 6), kdy všechny rozhodnutí provádí expertní systém náhodně, tak hráči dosáhnou svého původního skóre.

Pro psaní báze znalostí jsem využíval jména proměnných začínajících písmeny X, Y a Z. V pravidlech samotných pak proměnné začínající jiným písmenem. Tato písmena byla zvolena, protože vestavěné predikáty mPrologu je také využívají.

7.1 Způsob uložení informací o hře

Hra Taroky má mnoho možností a variant hraní, jak již bylo řečeno v kapitole 2. Expertní systém byl navržen natolik obecně, aby umožňoval zvolit programátorovi vhodný způsob ukládání informací o hře. Z těchto dvou důvodů a samozřejmě s ohledem na možnosti použitého interpretu MProlog musím rozhodnout, jakým způsobem ukládat data o hře. Jedná se o návrh struktury báze znalostí a pravidel typu set, které bázi znalostí naplní daty plynoucími z jednotlivých her.

V této fázi má hráč řízený přístrojem proti lidskému hráči bezesporu výhodu v tom, co všechno si je schopen bezchybně zapamatovat. Proto je třeba této části návrhu věnovat zvýšenou pozornost a navrhnout takové datové struktury, které nejlépe zvládnou uchovat informaci o hře. Mezi hlavní požadavky na tyto struktury přitom patří paměťová nenáročnost a schopnost na základě nich rychle a bezchybně určit platnost podmínek expertního systému.

Otázkou však zůstává na základě, jakých informací se skuteční hráči taroků rozhodují. Případně jaké další informace by chtěli při rozhodování mít, ale nejsou schopni si zapamatovat příslušná data k odvození těchto informací. Pravidla pro chování hráče expertního systému jsou založena na napodobování rozhodování skutečných hráčů. Tato pravidla se snaží co nejvěrněji napodobit jejich dobrá rozhodnutí. Skutečný hráč se většinou rozhoduje podle právě odhozených karet, jsou čtyři a jsou vidět, tedy nemusí si nic pamatovat. Dále se rozhodují podle přibližných počtů karet v jednotlivých barvách, které byly použity v minulých kolech hry. Z těchto počtů jednoduchým

způsobem určuje pravděpodobnost, zda dané kolo bude bez trumfů, popřípadě který hráč bude moct v daném kole nahrát spoluhráči shozením lépe bodované karty. Pokud to není zřejmé z druhu hry nebo zatím odhozených karet, tak si hráč v průběhu celé hry utváří představu, kdo s kým hraje. Při kvalitnějších hrách na vyšší úrovni hráči někdy dokonce nahrají soupeři, aby jej zmátli a on jim pak nahrál zpět ještě více bodů. Tento a další způsoby hraní bohužel nebylo možno zkoumat, protože hráči neradi prozrazují své triky, které rozvinuli dlouholetým hraním hry.

Je zřejmá nutnost uložit si hlášení jednotlivých hráčů, případně na základě hlášení vytvořit nebo nějak dotvořit pravidla pro kombinování možností o tom, co má kdo na ruce. Dále se pro samotné hraní nabízejí dvě možnosti přístupu, buď ukládat kompletní informaci o hře, kdo v kterém kole použil jakou kartu, nebo ukládat pouze souhrnné informace o počtech karet v jednotlivých barvách. Kombinací obou přístupů vzniká asi nejpřijatelnější řešení. Ukládám všechny použité karty, aby se hráč počítače mohl po odehráni dostatečného množství karet rozhodovat podle potenciálních možností karet soupeřů a případných spoluhráčů. Zároveň také ukládám souhrnné informace o počtech odehraných karet v jednotlivých barvách a další pomocné údaje, které lze sice zjistit z předchozích dat, ale jejich odvození by bylo výpočetně i paměťově náročné. Vytvořil jsem definici pravidel a datových struktur nazvanou *Logger*, která slouží k ukládání informací o hře.

7.2 Pravidla a datové struktury Logger

Logger je soubor pravidel definujících chování hráče expertního systému. Tento soubor neobsahuje žádná pravidla pro rozhodování herních situací. Obsahuje předpřipravené jednoduché termy s informacemi o stavu hry a příslušná pravidla, která tyto termy udržují ve stavu odpovídajícím realitě. Dále obsahuje některé složitější termy, které ukazují jak na základě jednoduchých termů lze odvodit pokročilejší informace. Soubor Logger slouží jako základ pro dále popisované soubory pravidel a může případně posloužit jako základ pro rozhodovací pravidla vytvořená třetí stranou. Proto je vhodné pochopit, co přesně jednotlivé termy znamenají a jak je zamýšleno jejich budoucí použití případně, která data jsou v tomto souboru zanedbávána a mohly by se hodit při vývoji dalších konfiguračních souborů pro definici inteligence úspěšnějších hráčů.

Term V bázi znalostí	**Stručný význam termu**
`mam(CARD).`	karta, kterou mám na ruce
`pouzil(TURN,PLAYER,CARD,ORDER).`	hráč `PLAYER` odložil v kole `TURN` v pořadí `ORDER` kartu `CARD`
`bonus(PLAYER,BONUS).`	hráč `PLAYER` hlásil hlášku `BONUS`
`bit(PLAYER,BIT).`	`BIT` je vydražena hra hráčem `PLAYER`
`turn(TURN).`	`TURN` poradí kola, ve kterém se hra nachází.
`turnOrder(ORDER).`	`ORDER` odpovídá počtu karet shozeném v daném kole.
`pocetKaret(POCET).`	`POCET` je počet karet, které mám na ruce
`mamOdlozit(CARD).`	`CARD` je karta, kterou mám odložit v úvodu hry.
`hodnota(CARD, VAL).`	`VAL` je ocenění karty `CARD` pro závěrečné sčítání
`hrac(PLAYER, BARVA, MIN, MAX).`	hráč `PLAYER` má na ruce karet v barvě `BARVA`

	minimálně `MIN` a maximálně `MAX`
`povinnost(POV,CARDVAL).`	mapuje na karty pro tvorbu týmů při jednotlivých verzích první a druhé povinnosti `POV` povinnost `CARDVAL` je hodnota taroku pro partnera teto povinnosti.
`team(X,Y).`	Test zda hráč X a Y jsou ve stejném týmu.

Jednoduché termy `mam(CARD)`, `bonus(PLAYER,BONUS)`, `bit(PLAYER,BIT)` jsou základní datovou strukturou. Pro každou kartu, kterou hráč dostane pomocí bloku SETCARD je přidán další elementární term `mam(CARD)`. Podobně pro každé zavazující či prozrazující hlášení je kromě jiného přidán jeden elementární term `bonus(PLAYER, BONUS)`. Dražba hry je pak ukládaná v `bit(PLAYER, BIT)`. Volná proměnná CARD je vždy nahrazena konkrétní kartou. Karta je v expertním systému chápána jako dvouprvkový seznam, kde první hodnotou je barva karty a druhou její pořadí v rámci dané barvy. Toto číslování vychází z definice expertního systému v kapitole 6. Termy `mam(CARD)` jsou mazány buď v bloku SETTURN, kde je oznámeno použití dané karty, nebo v bloku GETTURN jedná-li se o odložení karet v úvodu hry. Více o tomto je popsáno v části věnované `mamOdlozit(CARD)`. `PLAYER` je číslo pozice hráče, `BONUS` je kód hlášení a `BIT` je kód hry stejně, jak je to popisováno v kapitole 6.

Termy `pouzil(TURN,PLAYER,CARD,ORDER)` jsou asi nedůležitější v celém systému. Zde jsou uloženy informace o tom, jaké karty byly shozeny v jednotlivých kolech. Na základě těchto bloků by bylo možno klidně odvozovat další uložené informace zejména o počtu karet v jednotlivých barvách pro každého hráče. `TURN` zde označuje kolo, ve kterém ona karta byla odhozena. `PLAYER` je pozice hráče, který kartu v daném kole použil. `CARD` je tradiční dvouprvkový seznam reprezentující kartu. `ORDER` je pořadí v rámci daného kola kolikátá byla karta shozena. Z těchto termů lze rekonstruovat kompletně hranou hru. Při vytváření a udržování obsahu tohoto termu je brán ohled i na pátou kartu při hraní hry Varšava. Tuto pátou kartu jakoby shazuje virtuální hráč číslo čtyři.

Term `turn(TURN)` je čítač inicializovaný na nulu pro odkládání karet v úvodu hry a následně obsahuje číslo kola, ve kterém se hra nachází. Jedná se o redundantní data odvoditelná nalezením maxima v `pouzil(TURN,PLAYER,CARD,ORDER)` a doplněném o podmínku přechodu na další kolo. Dotaz na aktuální tah je využíván tak často, že jeho odvozováním se i práce Logger zpomalila téměř trojnásobně a to jak v emulátoru, tak na reálném zařízení. Udržováním čítače aktuálního sice roste počet pravidel, ale zrychlení je znatelné.

Term `turnOrder(ORDER)` je čítač počtu karet v rámci daného kola. Tento čítač je již odvozovaný z `turn(TURN)` a `pouzil(TURN, PLAYER, CARD, ORDER)` nalezením maximálního `ORDER` v aktuálním kole. Protože kolo může obsahovat maximálně pět karet, je hledání maxima v tomto případě statické bez rekurze, čímž se ušetří výkon a paměťová náročnost.

```
turnOrder(5):-turn(Y), pouzil(Y,_,_,5).
turnOrder(4):-turn(Y), pouzil(Y,_,_,4).
turnOrder(3):-turn(Y), pouzil(Y,_,_,3).
turnOrder(2):-turn(Y), pouzil(Y,_,_,2).
turnOrder(1):-turn(Y), pouzil(Y,_,_,1).
turnOrder(0).
```

Zjištění kolik dané kolo obsahuje karet.

Čítač počtu karet které mám na ruce `pocetKaret(POCET)`, je aktualizován pokaždé, když se tato skutečnost změní. V Logger je využíván hlavně k rozhodnutí, zda v bloku GETTURN hraji kartu nebo jsem v úvodu hry a kartu pouze odkládám.

Term `mamOdlozit(CARD)` do `CARD` vkládá kartu, která má být expertním systémem odložena, pokud je úvodní fáze hry a odkládají se karty. V implementaci tohoto pravidla se Logger lehce liší od Empty. Logger odloží vždy kartu, která je nejlépe hodnocená, proto v testech proti hráči Empty vyhrává. U tohoto pravidla je pro správnou následující funkci nutné kontrolovat, zda daná karta může být odložena. Pokud by byla vrácena karta, která nelze odložit, Logger by tuto informaci žádným způsobem už nedokázal zjistit a kartu by si odebral s karet na ruce a upravil všechny relevantní čítače. Expertní systém by ve skutečnosti zjistil, že odvozená karta nemůže být vrácena a vybral by první kartu z ruky, která být vrácena může. Toto chování je trochu problém návrhu expertního systému.

```
mamOdlozit(XCARD):-odkladase(XCARD),hodnota(XCARD, 4).
mamOdlozit(XCARD):-odkladase(XCARD),hodnota(XCARD, 3).
mamOdlozit(XCARD):-odkladase(XCARD),hodnota(XCARD, 2).
mamOdlozit(XCARD):-odkladase(XCARD),hodnota(XCARD, 1).
odkladase(XCARD):-mam(XCARD),pocetKaret(XPOCET),XPOCET>12,jdeOdlozit(XCARD).
jdeOdlozit([XBARVA, XHODNOTA]):-XBARVA>=0, XBARVA<4, hodnota([XBARVA, XHODNOTA],X), X<5.
jdeOdlozit(XCARD):-not(mam([0,_])),not(mam([1,_])),not(mam([2,_])),not(mam([3,_]),
  hodnota(XCARD,X), X<5.
```

Implementace `mamOdlozit(CARD)` v Logger

Term `hodnota(CARD, VAL)` slouží k zjišťování hodnoty karty CARD při závěrečném hodnocení hry. Cílem hry je většinou sebrat co nejvíc bodů, proto při rozhodování, zda přebíjet nebo se zdržet, je informace o hodnotách shozených karet jednou z klíčových.

Dalším důležitým termem umožňujícím rozhodovaní podobné tomu, které používají skuteční hráči, je `hrac(PLAYER, BARVA, MIN, MAX)`. Je v něm v průběhu hry uložena informace o počtech karet v jednotlivých barvách na ruce každého hráče. Tato informace je upravována v závislosti na hlášení hlášek Barvy, Barvičky, Taroky a Taročky, kterými hráč prozrazuje informací o svých kartách. V případě, kdy hráči dojdou karty některé barvy jsou čítače patřičně upraveny.

Dalším důležitým odvozovacím termem je `team(X,Y)`. Tento výrok je splnitelný, pokud z uplynulé hry jasně vyplývá, že hráči X a Y hrají spolu v jednom týmu. Zejména je to důkladně řešeno pro první a druhou povinnost, ale platí i pro ostatní varianty hry. Platí, že vždy je každý hráč v týmu sám se sebou.

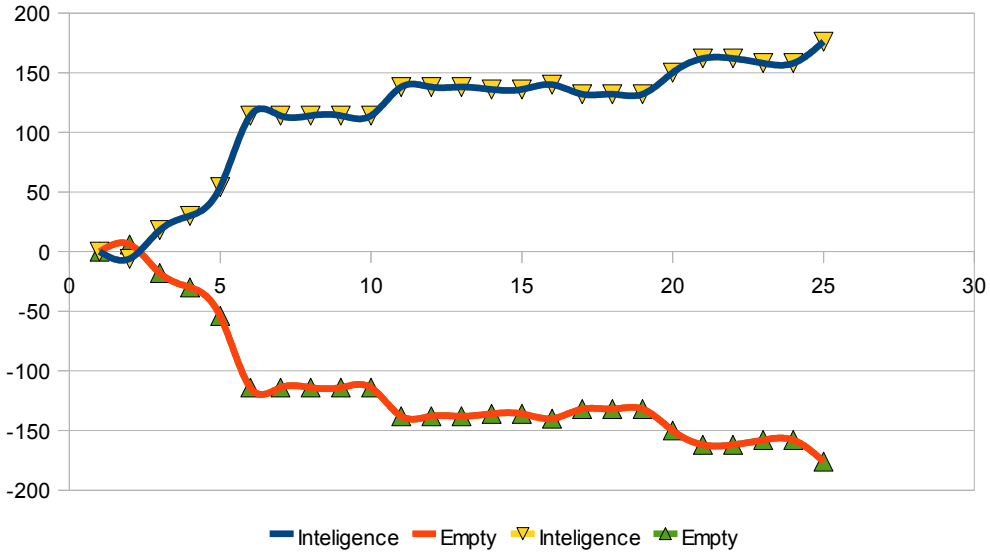
```
team(X,Y):-teamPP(X,Y).
team(X,Y):-playerPos(Z1), playerPos(Z2),not(eq(Z1,X)), not(eq(Z2,X)), not(eq(Z1,Y)),
  not(eq(Z2,Y)), not(eq(Z1,Z2)), teamPP(Z1,Z2).
team(X,Y):-bit(Z,XGAME), XGAME>10, not(eq(Z,X)), not(eq(Z,Y)).
teamPP(X,X).
teamPP(X,Y):-teamP(X,Y).
teamPP(X,Y):-teamP(Y,X).
teamP(XP,YP):- bit(YP,XPOV), povinnost(XPOV,XH), pouzil(_,XP,[4,XH],_).
teamP(XP,0):- bit(XP,XPOV), povinnost(XPOV,XH), mam([4,XH]).
```

Implementace testu zda dva hráči jsou spolu v jednom týmu.

7.3 První verze inteligence

Cílem je vytvoření souboru pravidel, které v nasazení proti lidskému hráči nebudou dělat znatelné strategické chyby. Zejména se zde bude jednat o rozhodování, kdy přebíjet a kdy nedělat nic. Popřípadě kdy nahrávat a kdy nenahrávat. Jedná se o pasivního hráče, který nedraží hry, ale pouze hraje hru s ostatními hráči.

První inteligenci byl dodán term `turnWinCard(Y)` pro zjištění, která karta bere kolo a možnost porovnávat karty na přebíjení `isBigger([XC,XR],[YC,YR])`. Pomocí nichž bylo vytvořeno první pravidlo pro rozhodování, které by se volně dalo interpretovat asi takto: „Mám-li krále, který přebije všechny karty v daném kole a je možné, že ostatní hráči nebudou přebíjet tarokem tak tohoto, krále použiji.“



obrázek 12: počet bodů hráčů v závislosti na počtu odehraných her.

Empty jsou hráči bez pravidel.

Inteligence jsou hráči s výše popsaným pravidlem pro odhazování králů

Toto pravidlo se zabývá pouze používáním králů. Dalším pravidlem jsem se snažil vhodným způsobem definovat nahrávání spoluhráčům shozením lépe bodovaných karet. Pravidlo zní takto: „Pokud talon bere spoluhráč a asi nebude přebit protihráčem tak shodit co nejlépe bodovanou kartu.“ Zde již zlepšení není tak znatelné, protože případů, kdy tato situace nastává, je podstatně méně. Z experimentování vyplynulo užití tohoto pravidla zhruba v deseti procentech odehraných her a vždy vedlo k lepším výsledkům tohoto hráče.

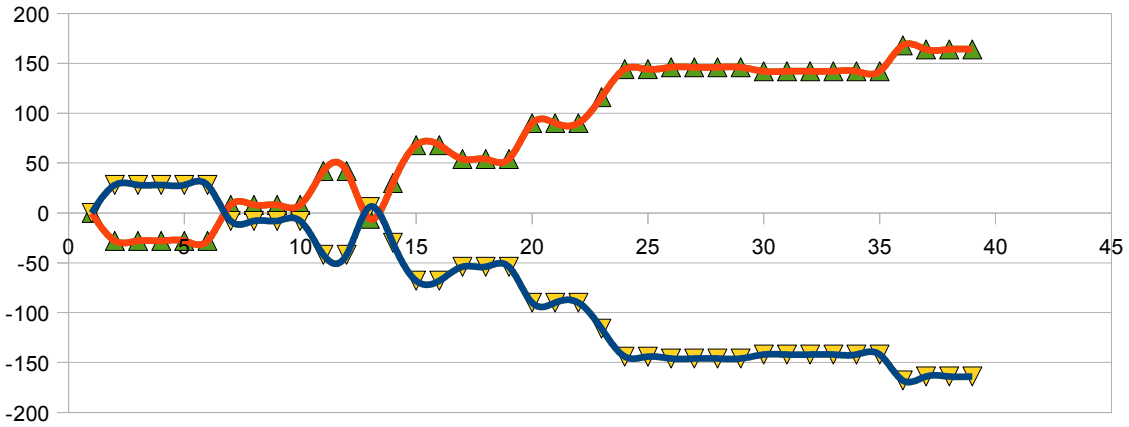
Podmínky tohoto pravidla

```
getTurn(X):-bereTalonSpoluhrac(), mamHodnotnou(X), canPlay(X),turn(XTURN),
              pouzil(XTURN,_,[XB,_],1), turnOrder(XORDER), nasledniciMajiBarvu(XORDER, XB).
bereTalonSpoluhrac():- team(X,0),turnWinCard(Y), pouzil(_,X,Y,_).
mamHodnotnou(X):- mam(X), hodnota(X,5).
mamHodnotnou(X):- mam(X), hodnota(X,4).
mamHodnotnou(X):- mam(X), hodnota(X,3).
```

```
mamHodnotnou(X):- mam(X), hodnota(X,2).
```

7.4 Inteligence pro různé typy her

Po optimalizacích první inteligence pro hraní normální hry jsem ji rozšířil o možnost hrát více druhů her. A tím získat větší množství bodů za hru, pokud má dobré karty. Nejprve jsem přidal pravidla pro volání hráče, se kterým je hrána první či druhá povinnost tak, aby hráč nehrál zbytečné sám se sebou, pokud to není nutné. Z grafu na obrázku 13 je vidět, že toto pravidlo přináší i určité negativní skoky. K těmto dochází, pokud bylo skutečně lepší hrát sám se sebou první povinnost nebo ještě lépe hlásit některou vyšší variantu hry.



obrázek 13: Počet bodů v závislosti na počtu odehraných her pro novější verzi pravidel reprezentovanou červenou čárou a zelenými body

Další implementovaná pravidla se týkají druhé povinnosti případně hlášky Pagát. Nejprve pravidla, která rozhodují, kdy Pagát hlásit a kdy nikoli. Jakmile byla hotova pravidla pro hraní Pagáta, mohla být vyvíjená série proti pravidel pro konkurenci hlášení Pagát. Flek na Pagáta je udělován náhodně v závislosti na počtu taroků. Obdobným způsobem pak byla vyvinuta pravidla pro hraní hry Trojka Sólo a Valát. Vždy jsem konzultoval rozhodování dané situace se zkušeným hráčem. Následně jsem implementoval jeho způsob rozhodování a korigoval různé koeficienty tak, abych maximalizoval bodový zisk vzniklý přidáním pravidla. Nakonec byla hra rozšířena mezi hráče, kteří ji odzkoušeli, a na základě jejich připomínek byly dodány poslední pravidla. Jsou spíše velice konkrétního charakteru a opravující chování ve specifických situacích, kdy je skutečnému hráči jasné, co má hrát, ale ES to nebyl z báze znalostí schopen odvodit. Kompletní výsledný soubor pravidel je dostupný na přiloženém CD a je použit jako implicitní inteligence pro hraní mobilní hry Taroky.

8 Závěr

Cílem této diplomové práce bylo vytvořit hru umožňující hraní jak proti automatickým protihráčům, tak proti lidským protihráčům. Automatičtí protihráči jsou reprezentováni expertním systémem, který je touto prací přesně definován. Expertní systém je navržen a implementován tak, že báze znalostí a odvozovací pravidla je možno načíst ze souboru a hrát proti hráči, jehož chování si nadefinuje uživatel sám, případně jej stáhne.

Pro reprezentaci báze znalostí je v expertním systému použito zjednodušené verze jazyka Prolog. Je využívána externí knihovna interpretu Prologu nazvaná **MProlog**.

8.1 Výsledná implementace

Hra je plně funkční a hratelná, byla testována na mobilních telefonech: NOKIA N96, NOKIA 9300, a levnějším typu telefonu Motorola U9. Aplikace na všech spolehlivě funguje, pouze Motorola nedisponuje potřebným API pro práci se souborovým systémem, proto není u tohoto telefonu možno využívat aktualizace pravidel expertního systému.

Aplikace lehce překračuje paměťová omezení specifikovaná pro J2ME. Jedná se o grafickou aplikaci a je zde potřeba obrázků. Ale hlavně se odvozování platnosti podmínek v interpretovaném jazyce mProlog. Podle *memory monitoru*, který je součástí *Wireless toolkit*, aplikace bez expertního potřebuje k běhu necelých 105kB paměti, z toho 51kB tvoří obrázek karet. Při použití expertního systému s poslední verzí inteligence je paměť emulátoru využívána až po svou mezní hodnotu 2MB. Následně je vidět spouštění *garbage collector*, který uvolní paměť tak, že je využíváno 400MB, a poté opět využívání rovnoměrné stoupá.

Expertním systémem pro mobilní telefon se již zabývala bakalářská práce Tomáše Pumprly s názvem: „Karetní hra Mariáš pro mobilní zařízení“. Bohužel v rámci této práce nebylo rozhraní expertního systému ani jeho možnosti dostatečně popsáno. Je zde pouze zmínka o využívání expertního systému, díky kterému je možno vyměnit inteligenci bez nutnosti rekompilace.

Mezi další implementace hry Taroky patří stejnojmenný program od společnosti cPortal z roku 2005 určený pro stolní počítače. Porovnání úspěšnosti automatických hráčů z této hry a hráčů z mé hry pro mobilní zařízení by mohlo být zajímavé, bohužel jsem se k němu nedostal. Mezi hlavní výhody jejich uživatelského rozhraní patří možnost zobrazit historii hlášení jednotlivých hráčů a pohodlnější ovládání pomocí myši místo malých tlačítek mobilního telefonu. Další výhodou jejich řešení je možnost hraní proti hráčům on-line [21].

8.2 Možnosti rozšíření a budoucnost aplikace

Do budoucna by bylo dobré dodat možnost hrát s hráči přes bluetooth nebo po síti. Dalším vhodným rozšířením je odvození pravidel silně připomínající naučenou neuronovou síť, aby pro odvozování nebylo potřeba tolik paměti. Nebylo by špatné zkusit optimalizovat knihovnu mProlog nebo ji úplně nahradit jinou implementací. Rovněž se nabízí možnost pouvažovat nad možností využít expertního systému jako nápovědu při hraní reálné hry.

Literatura

[1] *Tarokový klub Praha* [online]. 2008 [cit. 2010-01-02]. Z historie hry Taroky. Dostupné z WWW: <http://taroky.z80.cz/historie.html>.

[2] *Wikipedia* [online]. 2010 [cit. 2010-01-01]. Taroky. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Taroky>.

[3] *CARD GAMES* [online]. 2009 [cit. 2010-02-15]. Taroky. Dostupné z WWW: <http://www.pagat.com/tarot/taroky.html>.

[4] ZUZAŇÁK, J. *Platforma J2ME a mobilní Bluetooth* [online]. Brno : VUT, [2009]. 53 s. Opora k přednáškám. VUT, FIT. Dostupné z WWW: <http://www.fit.vutbr.cz/~izuzanak/www/tam/slides.pdf>

[5] *Java Community process* [online]. 2010 [cit. 2010-01-03]. Participation JCP Members. Dostupné z WWW: <http://jcp.org/en/participation/members>.

[6] PUMPRLA, T. *KARETNÍ HRA MARIÁŠ PRO MOBILNÍ ZAŘÍZENÍ*. Brno, 2007. 31 s. Bakalářská práce. VUT, FIT.

[7] *J2ME Building Blocks for Mobile Devices* [online]. Palo Alto : Sun Microsystems, 2000 [cit. 2010-01-22]. Dostupné z WWW: <http://java.sun.com/products/cldc/wp/KVMwp.pdf>.

[8] MAHMOUD, H. *Naučte se Java 2 Micro Edition*. Praha : Grada, 2002. 246 s.

[9] *JavaME* [online]. 2006 [cit. 2010-01-03]. CLDC 1.1. Dostupné z WWW: <http://java.sun.com/javame/reference/apis/jsr139/>.

[10] *The CLDC HotSpot™ Implementation Virtual Machine* [online]. [s.l.] : [s.n.], 2002 [cit. 2010-05-24]. Dostupné z WWW: <http://java.sun.com/products/cldc/wp/CLDC_HI_WhitePaper.pdf>.

[11] *Sun Developer Network* [online]. 2010 [cit. 2010-04-20]. What's New in MIDP 2.0. Dostupné z WWW: <http://java.sun.com/products/midp/whatsnew.html>.

[12] FEIGENBAUM, A., et al. *The Rise of the Expert Company*. London : Macmillan, 1988. 336 s.

[13] CELBOVÁ, I. *IKAROS* [online]. 1999 [cit. 2010-04-30]. Úvod do problematiky expertních systémů. Dostupné z WWW: <http://www.ikaros.cz/uvod-do-problematiky-expertnich-systemu>.

[14] DVOŘÁK, J. *Expertní systémy*. Brno : VUT, 2004. 92 s.

[15] LUGER, R., et al. *LinkAI algorithms, data structures, and idioms in Prolog, Lisp, and Java*. Boston : Pearson Education, 2009. 437 s.

[16] MAŘÍK, V., et al. *Umělá inteligence*(2). Praha : Academia, 1997. 373 s.

[17] *Stanford Logic Group* [online]. 1998 [cit. 2010-05-24]. Knowledge Interchange Format. Dostupné z WWW: <http://logic.stanford.edu/kif/dpans.html>.

[18] *Stanford Logic Group* [online]. 2005 [cit. 2010-05-24]. Ontolingua. Dostupné z WWW: <http://logic.stanford.edu/kif/dpans.html>.

[19] LÁNÍK, A. *TAM-J2ME pokracovan* [online]. Brno : VUT, [2009]. 38 s. Opora k přednáškám.. VUT, FIT. Dostupné z WWW: <https://www.fit.vutbr.cz/study/courses/TAM/private/lect/TAM-J2ME-II.pdf>.

[20] *3APL-M* [online]. 2005 [cit. 2010-05-24]. MProlog: Lightweight PROLOG Engine. Dostupné z WWW: <http://www.cs.uu.nl/3apl-m/mprolog.html>.

[21] *Instaluj* [online]. 2005 [cit. 2010-05-24]. Taroky. Dostupné z WWW: <http://www.instaluj.cz/taroky>.

Seznam příloh

Příloha 1. CD Obsahující:

- Zdrojové soubory aplikace
- Soubory definující jednotlivé diskutované verze expertních systémů
- Spustitelnou verzi aplikace pro mobilní zařízení nebo pro emulátor (.jar + .jad)
- Programovou dokumentaci
- Elektronickou verzi tohoto dokumentu
- Návod jak aplikaci spustit v emulátoru